תורת החישוביות - סיכומי הרצאות

גדי אלכסנדרוביץ'

תוכן עניינים

1 מבוא 2
2 מכונת טיורינג 3
2.1 המודל הבסיסי 3
2.1.1 מבוא 3
2.1.2 התיאור האינטואיטיבי של מכונת טיורינג 3
2.1.3 התיאור הפורמלי של מכונת טיורינג 4
2.1.4 דוגמא: מכונה שמזיזה את הקלט 5
2.1.5 דוגמא: מכונה שמשכפלת את הקלט 6
2.2 שקילות מודלים 7
2.2.1 מבוא 7
2.2.2 דוגמא: שקילות למכונה "מהירה" 7
2.2.3 דוגמא: שקילות למכונה דו־סרטית 8
2.2.4 מודל ה־RAM 9
2.2.5 התזה של צ'רץ' וטיורינג 10
2.3 מכונת טיורינג אוניברסלית 10
2.3.1 מבוא 10
2.3.2 קידוד 11
2.3.3 סימולציה 12
2.4 משפט הרקורסיה של קלייני 14
3 בעיות לא כריעות 15
3.1 בעיות הכרעה של שפות 15
3.2 רדוקציות 19
3.2.1 הגדרה 19
3.2.2 דוגמאות 19
3.2.3 משפט הרדוקציה 21
3.3 משפט רייס 22
3.4 הרצה מבוקרת 24
3.5 חישוב פונקציות 25
3.6 בעיות זיהוי וחיפוש של יחסים 27
3.7 סיבוכיות קולמוגורוב 28
4 מבוא לתורת הסיבוכיות 29
4.1 הגדרת חישוב יעיל 29
4.2 המחלקה NP 32
4.2.1 מבוא 32
4.2.2 הגדרה פורמלית 32
4.2.3 דוגמאות 33
4.2.4 הגדרה אלטרנטיבית - מכונות אי־דטרמיניסטיות 34
4.3 שאלת P=NP 35
5 בעיות NP־שלמות 37
5.1 מבוא 37

5.2 דוגמאות ראשונות לשפות NP־שלמות . . . 39
5.2.1 השפות SAT ו־k-SAT . . . 39
5.2.2 השפה Vertex Cover . . . 41
5.2.3 השפה Hitting Set ("קבוצה מייצגת") . . . 43
5.2.4 השפה Set Cover ("כיסוי בקבוצות") . . . 43
5.2.5 השפה IP01 (תכנון לינארי 01 בשלמים) . . . 44
5.3 הוכחות ישירות לכך ששפות הן NP־שלמות . . . 45
5.3.1 השפה Bounded Halting . . . 45
5.3.2 משפט קוק־לוין . . . 46
5.4 דוגמאות מתקדמות לשפות NP־שלמות . . . 49
5.4.1 בעיית סכום תת־הקבוצה (Subset Sum) . . . 49
5.4.2 בעיית החלוקה Partition . . . 50
5.4.3 בעיית החלוקה לתאים (Bin Packing) . . . 51
5.4.4 בעיות של גרפים המילטוניים . . . 51
6 נושאים נוספים . . . 54
6.1 אלגוריתמי קירוב . . . 54
6.1.1 הגדרה . . . 54
6.1.2 אלגוריתמי קירוב קונקרטיים . . . 55
6.1.3 קושי לקירוב של בעיות . . . 57
6.2 הוכחה בלכסון לקיום שפה $L \in R \backslash P$ . . . 59
6.3 סיבוכיות זיכרון . . . 59
6.3.1 הגדרה ותוצאות בסיסיות . . . 59
6.3.2 גרף הקונפיגורציות של מכונה . . . 60
6.3.3 השפה TQBF ומחלקת השפות ה־PSPACE־שלמות . . . 61

1 מבוא

תורת החישוביות היא הענף במדעי המחשב העוסק במגבלות התיאורטיות של מושג **החישוב**. בבסיסה, תורת החישוביות עוסקת בהגדרה פורמלית של חישובים, מציאת הקשרים בין הגדרות שונות, ומציאת המגבלות האינהרנטיות שקיימות על חישובים. ענף קרוב לתורת החישוביות הוא **תורת הסיבוכיות** שעוסקת במגבלות של חישובים שכפופים לאילוצי משאבים כלשהם (מוגבלים בזמן, בזיכרון, בכמות הביטים שניתן לתקשר בין שני מחשבים נפרדים וכדומה). בקורס זה נעסוק בהיכרות עם הבסיס של שני תחומים אלו.

היסטורית, תורת החישוביות התפתחה בשנות ה־30 של המאה ה־20, כתוצאה מהתפתחויות בתחומי הלוגיקה המתמטית. מוטיבציה עיקרית להתפתחות התחום ניתנה בידי המתמטיקאי דויד הילברט, שהציב "אתגר" בפני העולם המתמטי: למצוא אלגוריתם אשר מכריע, לכל טענה מתמטית, האם היא נכונה או שאינה נכונה. בעיה זו, שנקראה בגרמנית Entscheidungsproblem ("בעיית הכרעה"), התגלתה כבלתי ניתנת לפתרון ־ דהיינו, לא קיים אלגוריתם שמסוגל לבצע אותה. הדים לחוסר היכולת הזה אפשר למצוא כבר ב**משפטי אי השלמות של גדל** מתחילת שנות ה־30; אולם על מנת להשתמש ברעיונותיו של גדל כדי להוכיח כי האתגר של הילברט אינו פתיר, נדרשו המתמטיקאים למצוא מודל מתמטי פורמלי למושג הכללי של **חישוב**. מודל שכזה הוצע חלקית בידי גדל עצמו בעת הוכחת משפטי אי השלמות ("פונקציות רקורסיביות"), ומודל מרכזי נוסף ("תחשיב למבדא") הוצע בידי אלונזו צ'רץ', שהשתמש בו כדי להוכיח שבעיית ההכרעה של הילברט היא בלתי פתירה, אולם המודל המפורסם ביותר הוצע בידי אלן טיורינג במאמר משנת 1936 "On Computable Numbers, with an Application to the Entscheidungsproblem") שבו תיאר טיורינג את המודל שאנחנו מכנים כיום על שמו, **מכונת טיורינג**.

תורת הסיבוכיות החלה להתפתח בשנות ה־70 של המאה ה־20 אם כי ניתן למצוא הדים לה במכתב ששלח קורט גדל (שוב הוא!) לג'ון פון־נוימן בשנת 1956 אך אבד עקב מחלתו של פון־נוימן. בתורה זו עוסקים במודלים חישוביים הנוצרים תחת מגבלות חישוביות אלו ואחרות (למשל, מגבלות על זמן ריצה או על כמות הזיכרון שבה ניתן להשתמש), ובמודלים מורכבים יותר של חישוב (למשל חישוב הסתברותי, חישוב קוונטי והוכחות אינטראקטיביות), במחלקות הבעיות שניתן לפתור בעזרת מודלים אלו ובקשרים הרבים והמפתיעים ביניהן. השאלה הפתוחה המרכזית בתחום זה שעלתה כבר בשנות ה־70 היא שאלת $P = NP$ שניתן לתאר בצורה מקוצרת בתור "האם כל מה שקל לבדוק פתרונות עבורו גם קל לפתור?". בקורס זה נגיע לתיאור בעיית $P = NP$ ומספר בעיות שהיא רלוונטית עבורן, אך לא נגיע אל חלקה העיקרי של תורת הסיבוכיות; החומר שבו ניגע כאן מהווה מעין הקדמה אליה.

2 מכונת טיורינג

2.1 המודל הבסיסי

2.1.1 מבוא

על מנת לתאר אלגוריתמים אנו משתמשים בדרך כלל בשפה טבעית או בפסאודו־קוד. אלו הן שיטות תיאור לא פורמליות, אך די בהן לצרכים שלנו. מדוע, אם כן, נזקק אלן טיורינג להגדרה פורמלית של מודל חישובי? מכיוון שהוא לא רצה לתאר אלגוריתם קונקרטי, אלא ההפך ־ להראות שלבעיה מסויימת **לא קיים** אלגוריתם כלשהו שפותר אותה. לשם כך נדרשת הגדרה פורמלית של אלגוריתם שתאפשר לנו לטעון בביטחון שלא משנה כמה חכמים ויצירתיים נהיה בעתיד, שום אלגוריתם שנמציא לא יוכל לפתור את הבעיה.

האתגר שעמד בפני טיורינג הוא למצוא מודל שהוא

- **סביר**: כלומר, שהוא ניתן למימוש בפועל.
- **כללי**: כלומר, שכל חישוב שאנחנו מחשיבים כ"סביר" אכן ניתן לביצוע באמצעותו.
- **פשוט**: כלומר, שהגדרה פורמלית של חישוב באמצעותו תדרוש מספר קטן של מרכיבים.
- **אינטואיטיבי**: כלומר, שנוכל "להרגיש" את האופן שבו המודל קשור לתפיסה שיש לנו לגבי ביצוע חישובים.

המודל של טיורינג, שכונה על ידו "מכונת חישוב" ואנחנו פשוט מכנים **מכונת טיורינג** עונה על הדרישות הללו.

2.1.2 התיאור האינטואיטיבי של מכונת טיורינג

כאשר אדם ממוצע בזמננו מעוניין לבצע חישוב הוא נעזר במחשבון, אולם בזמנו של טיורינג טרם היו קיימים מחשבונים ולכן אדם ממוצע שנדרש לבצע חישוב היה נעזר בדף ועט. אם מתבוננים על אדם כזה בזמן ביצוע חישוב ומבלי להבין את פרטי החישוב, אפשר לראות שהוא עושה את הפעולות הבאות:

- **קורא** דברים ממקומות שונים על הנייר.
- **חושב** על הדברים שקרא על הנייר.
- **כותב** דברים במקומות שונים על הנייר.

אם נעקוב אחרי העיניים שלו, נראה כי בכל רגע נתון הן מתמקדות בחלק קטן מאוד של הנייר, וזזות אנה ואנה, בהתאם לדברים שקרא וחשב.

אנחנו לא יודעים מה מתרחש במוחו של האדם, אבל בהנחה שאין מדובר על מחשבון אנושי, כנראה שהוא עושה מעט מאוד. למשל, כאשר מבצעים חיבור ארוך על פי רוב המחשבה שלנו מסתכמת בחיבור שני מספרים מ־0 עד 9, ובאלגוריתם החיבור הארוך שאנחנו "מריצים". רוב יכולותיו המופלאות של המוח האנושי כלל אינן באות לידי ביטוי בהרצה זו של האלגוריתם. אינטואיטיבית, אפשר לחשוב על המוח שלנו כמורכב ממספר "מצבים תודעתיים" שאנחנו עוברים ביניהם תוך כדי החישוב (המצב שבו אנחנו רוצים לקרוא את הספרות הרלוונטיות הבאות; המצב שבו אנחנו זוכרים מה הספרות הללו ורוצים לחבר אותן בראש, וכן הלאה) והחישוב הוא אינטראקציה בין המצבים התודעתיים הללו, העין שקוראת דברים מהנייר, והיד שלנו שכותבת דברים על הנייר.

טיורינג הציע מכונה שפועלת בדיוק כך: היא כוללת אוסף סופי של "מצבים תודעתיים" (ה"מוח" של מכונה) שמחוברים אל מנגנון שכולל "ראש קורא/כותב" שמונח מעל סרט חד־ממדי שמחולק לתאים. בכל תא ניתן לכתוב סימן כלשהו מתוך קבוצה סופית של סימנים. הסרט עצמו אינו מוגבל; אנחנו לא מניחים שקיים לו קצה, כשם שהאדם בסיפור שלנו תמיד יכול לקחת עוד דף אם יזדקק לו. בפועל משאבי היקום מוגבלים ולא ניתן להניח שהסרט אינו מוגבל, אבל כזכור, אנו עוסקים בשאלה מה אלגוריתמים לא יכולים לבצע **בכלל**, בלי תלות במחסור במשאבי חישוב כמו נייר כתיבה, כך שאנו מניחים את ההנחה המקילה על הסרט.

המכונה פועלת בצעדים בדידים. בכל צעד, המכונה מצויה ב"מצב תודעתי" אחד ספציפי, ומתבוננת על תא ספציפי של הסרט. שני פריטי המידע הללו ־ המצב התודעתי של המכונה ותוכן התא בסרט שעליו היא מסתכלת ־ קובעים באופן חד־משמעי את הצעד הבא שלה. הצעד הבא כולל שלושה דברים: מעבר ל"מצב תודעתי" כלשהו (ניתן להישאר במצב הנוכחי), כתיבת תו כלשהו על הסרט במקום התו הקיים (ניתן להשאיר את התו הנוכחי), והזזה של הראש קורא/כותב לכל היותר צעד אחד ימינה או שמאלה (ניתן להישאר במקום הנוכחי).

בשלב מסוים המכונה יכולה להחליט ש"הספיק לה" **ולעצור**. אחרי שהיא עוצרת, הפלט של החישוב שלה יכול לכלול את כל המידע שנכתב על הסרט, אבל מטעמי פשטות ננקוט בגישה שונה מעט ־ הפלט הוא רק מה שנמצא בין הראש של המכונה ובין תחילת הסרט.

המכונה עלולה גם לא לעצור כלל, אם היא לא נכנסת אף פעם למצב התודעתי של "לעצור"; במקרה זה, הפלט שלה אינו מוגדר.

2.1.3 התיאור הפורמלי של מכונת טיורינג

הגדרה 1.2 מכונת טיורינג (מ"ט) היא שביעייה $M=(Q,q_0,F,\Gamma,\Sigma,\flat,\delta)$ כך ש:

- Q היא קבוצה **סופית** לא ריקה כלשהי. אבריה של Q מכונים **מצבים**.
- $q_0\in Q$ נקרא **המצב ההתחלתי**.
- $F\subseteq Q$ היא קבוצה שאבריה נקראים **מצבים סופיים**.
- Γ היא קבוצה סופית לא ריקה כלשהי כך ש־$Q\cap\Gamma=\emptyset$. Γ נקראת **א"ב העבודה**.
- $\Sigma\subsetneq\Gamma$ היא תת־קבוצה ממש של Γ שנקראת **א"ב הקלט**.
- $\flat\in\Gamma\backslash\Sigma$ נקרא "תו ריק" (blank) או "רווח".
- $\delta:(Q\backslash F)\times\Gamma\to Q\times\Gamma\times\{L,R,S\}$ נקראת **פונקציית המעברים**.

הגדרנו מכונת טיורינג וכעת נגדיר מהו חישוב שהמכונה מבצעת. לשם כך נזדקק למושג שמתאר "מצב רגעי" של החישוב.

הגדרה 2.2 קונפיגורציה ("מצב רגעי") של מ"ט M היא שלשה $C=(\alpha,q,i)$ כך ש:

- $q\in Q$ הוא מצב של M הנקרא **המצב הנוכחי** של החישוב.
- $\alpha\in\Gamma^*$ היא מחרוזת סופית של אברי Γ הנקראת **תוכן הסרט**.
- $i\in\mathbb{N}$ הוא מספר טבעי הנקרא **מיקום הראש**.

בנוסף, **הקונפיגורציה ההתחלתית** בחישוב של המ"ט M על הקלט x היא הקונפיגורציה $(x,q_0,0)$. **קונפיגורציה סופית** היא כל קונפיגורציה (α,q,i) עבורה $q\in F$.

אפשר לחשוב על חישוב בתור סדרה של מעברים בין קונפיגורציות שונות, בהתאם לתוכן הסרט, מיקום הראש הקורא/כותב וה"תוכנית" שאותה המכונה מריצה ומקודדת בעזרת פונקציית המעברים δ:

הגדרה 3.2 **צעד חישוב** של מ"ט הוא מעבר מקונפיגורציה (α,q,i) אל קונפיגורציה (β,p,j). צעד חישוב הוא חוקי אם הוא תואם את פונקציית המעברים. כלומר, אם $\delta\left(q,\alpha\left[i\right]\right)=(p,\gamma,X)$ ומתקיים:

- הראש זז בהתאם ל־X, כלומר:
  - אם $X=R$ אז $j=i+1$
  - אם $X=L$ אז $j=\max\{0,i-1\}$ (כלומר, הראש אינו יכול ללכת שמאלה אם הוא בתחילת הסרט)
  - אם $X=S$ אז $j=i$
- התו ה־i של α הוחלף ב־γ, ואם הראש של המכונה חרג מגבולות תוכן הסרט הקיים, התא החדש הוא ריק. כלומר, אם נסמן $\alpha=\alpha_0\alpha_1\cdots\alpha_n$ אז קיימות שתי אפשרויות:
  - אם $i<n$ או $i=n$ אבל $X\neq R$ אז $\beta=\alpha_0\cdots\alpha_{i-1}\gamma\alpha_{i+1}\cdots\alpha_n$.
  - אם $i=n$ וגם $X=R$ אז $\beta=\alpha_0\cdots\alpha_{n-1}\gamma\flat$.

אם C,C' הן שתי קונפיגורציות, נסמן $C\vdash C'$ אם צעד חישוב מ־C מעביר אותנו אל C'. במקרה זה נאמר ש־C' היא **הקונפיגורציה העוקבת** של C.

כעת ניתן להגיע אל הגדרת חישוב:

הגדרה 4.2 **החישוב** (או **הריצה**) של מ"ט M על קלט x הוא סדרת קונפיגורציות $C_0, C_1, C_2, \ldots$ כך ש־

- C_0 היא הקונפיגורציה ההתחלתית של M על x.
- לכל i, אם קיימת קונפיגורציה C כך ש־$C_i \vdash C$, אז $C_{i+1} = C$.

אם קיים C_n כך ש־C_n קונפיגורציה סופית (ולכן לא קיימת לה קונפיגורציה עוקבת) אומרים שהחישוב **מסתיים** והמכונה **עוצרת**. במקרה שבו החישוב מסתיים בקונפיגורציה $C = (\alpha, q, i)$, **הפלט** של החישוב הוא $\alpha_0\alpha_1 \cdots \alpha_{i-1}$, כלומר כל התווים על הסרט שהם **משמאל** לראש. אם אין כאלו, הפלט הוא המילה הריקה ε.

אם החישוב של M על x הוא אינסופי, אומרים ש־M **לא עוצרת** על x, והפלט של המכונה על x **אינו מוגדר**.

בהגדרה זו, מכונת טיורינג היא אמצעי חישוב שמקבל קלט (תוכן הסרט בתחילת החישוב) ומוציאה פלט (תוכן הסרט משמאל לראש בסוף החישוב). זה מתאר פונקציה:

הגדרה 5.2 בהינתן מ"ט M, הפונקציה ש־M מחשבת היא הפונקציה $f_M : \Sigma^* \to \Gamma^*$ המוגדרת באופן הבא:

- אם הפלט של M על x הוא y, אז $f_M(x) = y$.
- אם הפלט של M על x אינו מוגדר, אז $f_M(x)$ אינה מוגדרת (לעתים מסמנים זאת $f_M(x) = \perp$).

פורמלית, פונקציה $f_M : \Sigma^* \to \Gamma^*$ צריכה על פי הגדרה להיות מוגדרת לכל התחום שלה, כלומר לכל Σ^*. בפועל כדי לפשט את הסימונים אנו לא משנים את התחום גם כאשר f_M אינה מוגדרת על כולו (מי שזקוקים לפורמליות מלאה יכולים להגדיר $f_M : \Sigma^* \to \Gamma^* \cup \{\perp\}$ כדי ש־f_M תהיה מוגדרת פורמלית על כל קלט). על מנת להדגיש שפונקציה מוגדרת על כל התחום שלה, נשתמש בשם מיוחד:

הגדרה 6.2 פונקציה $f : \Sigma^* \to \Gamma^*$ תיקרא **מלאה** אם היא מוגדרת לכל קלט.

2.1.4 דוגמא: מכונה שמזיזה את הקלט

נבנה מ"ט M שמחשבת את הפונקציה $f(x) = 0x$ עבור $x \in \{0,1\}^*$. כלומר ־ הפונקציה לוקחת את הקלט שעל הסרט, מזיזה אותו צעד אחד ימינה וכותבת 0 במקום שהתפנה בתחילת הסרט.

האלגוריתם שמאחורי מכונה שכזו הוא פשוט: בכל צעד יש לזכור את התו שמופיע כרגע על הסרט ולכתוב במקומו את התו שהיה קודם משמאלו (או 0, אם לא היה תו קודם משמאלו). אז מזיזים את הראש ימינה וחוזרים על הפעולה. החישוב מסתיים כאשר נתקלים לראשונה ב־$\flat$, שפירושו שהגענו אל קצה הקלט x.

נשתמש במצבים כדי לזכור מה התו שאנחנו צריכים לכתוב על הסרט: אם אנחנו צריכים לכתוב 0 המצב יהיה q_0 ואם אנחנו צריכים לכתוב 1 המצב יהיה q_1. באופן ממוזל ואקראי לחלוטין יוצא שבתחילת ריצת המכונה, כאשר היא במצב ההתחלתי שנהוג לסמן ב־q_0, התו שיש לכתוב על הסרט במקום הנוכחי הוא אכן 0.

בנוסף לכך נזדקק למצב סופי q_f שמציין את סוף הריצה. אם כן, במכונה $(Q, q_0, F, \Gamma, \Sigma, \flat, \delta)$ שלנו:

- $Q = \{q_0, q_1, q_f\}$
- $F = \{q_f\}$
- $\Gamma = \{0, 1, \flat\}$
- $\Sigma = \{0, 1\}$

את δ ניתן לתאר באמצעות טבלה:

	0	1	$\flat$
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$	$(q_f, 0, R)$
q_1	$(q_0, 1, R)$	$(q_1, 1, R)$	$(q_f, 1, R)$

ניתן להוכיח באינדוקציה את נכונות פעולת המכונה, אבל נוותר על כך כאן.

2.1.5 דוגמא: מכונה שמשכפלת את הקלט

עבור Σ כלשהו נבנה מ"ט M שמחשבת את הפונקציה $f\left(x\right)=xx$ עבור $x\in\Sigma^{*}$. לצורך כך נפעל בצורה הבאה:

- לכל אות בקלט המקורי, נקרא את האות, נלך ימינה עד למופע הראשון של $\flat$ ונכתוב את האות במקום זה.
- נחזור שמאלה עד שנגיע אל האות הראשונה שטרם טופלה בצורה הזו.

נשאלת השאלה כיצד ניתן לדעת אילו אותיות כבר טופלו ואילו לא. התשובה היא שנסמן אותן בסימון מיוחד. לכל תו $\sigma\in\Sigma$ נשתמש גם בתו σ' כאשר הסימון $'$ על תו מתאר שהוא כבר טופל. בצורה הזו, כדי למצוא את האות הראשונה שטרם טופלה יש ללכת שמאלה עד אשר מופיע תו עם $'$ (זוהי האות האחרונה **שכבר טופלה**), ואז ללכת צעד אחד ימינה כדי להגיע לאות שטרם טופלה.

בעיה נוספת היא שאנו כל הזמן מייצרים תווים חדשים על הסרט - איך לא נתבלבל ונתחיל לשכפל גם אותם? יש לכך שני פתרונות אפשריים:

- אפשר לשים תו מיוחד שמפריד בין x ובין החלק המשוכפל שלו, ואחרי שסיימנו את השכפול, להזיז צעד אחד שמאלה את כל החלק המשוכפל (כפי שראינו כיצד להזיז ימינה מחרוזת בדוגמא הקודמת).
- אפשר להשתמש בתווים מתוייגים מסוג נוסף כדי לסמן את התווים המשוכפלים.

ננקוט בגישה השניה: לכל $\sigma\in\Sigma$ נשתמש בתו σ'' כדי לסמן אותיות שנכתבו על הסרט על ידינו, וכך לא נשכפל אותן בטעות. נדע שסיימנו את החישוב אם מימין לתו עם תג $'$ יופיע תו עם תגיים $''$.

כאשר זיהינו שהחישוב הסתיים, נלך אל הקצה הימני שבו השתמשנו של הסרט (עד שמופיע $\flat$) ואז נתחיל לנוע שמאלה תוך שאנו מסירים את התגים מכל תו שאנו רואים.

כיצד נזהה מתי הגענו לקצה השמאלי של הסרט? אם אנחנו בקצה השמאלי, תנועה שמאלה לא תעשה דבר אלא תשאיר אותנו במקומנו; מכיון שאנחנו מסירים תגים בכל צעד, זה יותיר אותנו בתו שאינו מתוייג, ונוכל להשתמש בכך כדי לזהות שסיימנו. בשלב זה נחזור שוב אל הקצה הימני של הסרט ונסיים.

המצבים שלנו יהיו:

- q_0 - המצב ההתחלתי שמשמעותו "תייג את האות הנוכחית בקלט וזכור שאתה צריך לשכפל אותה".
- לכל $\sigma\in\Sigma$: מצב q_{σ}^{R} שמשמעותו "לך ימינה עד קצה הסרט וכתוב שם σ".
- q_1 - מצב שמשמעותו "לך שמאלה עד לתו המתויג הראשון ואז לך ימינה".
- q_2 - מצב שמשמעותו "השכפול הסתיים; לך לקצה הימני של הסרט".
- q_3 - מצב שמשמעותו "לך שמאלה עד קצה הסרט והסר את התגים שבהם אתה נתקל".
- q_4 - מצב שמשמעותו "שלב הסרת התגים הסתיים; לך לקצה הימני של הסרט".
- q_f - מצב סופי.

כלומר:

- $Q=\left\{ q_0,q_1,q_2,q_3,q_4,q_f\right\} \cup\left\{ q_{\sigma}^{R}\mid\sigma\in\Sigma\right\}$
- $F=\left\{ q_f\right\}$
- $\Gamma=\Sigma\cup\left\{ \sigma'\mid\sigma\in\Sigma\right\} \cup\left\{ \sigma''\mid\sigma\in\Sigma\right\} \cup\left\{ \flat\right\}$

ופונקציית המעברים δ מוגדרת על ידי

	τ	τ'	τ''	$\flat$				
q_0	$\left(q_{\tau}^{R},\tau',R\right)$		$\left(q_2,\tau'',S\right)$	$\left(q_f,\flat,S\right)$				
q_{σ}^{R}	$\left(q_{\sigma}^{R},\tau,R\right)$		$\left(q_{\sigma}^{R},\tau'',R\right)$	$\left(q_1,\sigma'',L\right)$				
q_1	$\left(q_1,\tau,L\right)$	$\left(q_0,\tau',R\right)$						
q_2		$\left(q_2,\tau',R\right)$	$\left(q_2,\tau'',R\right)$	$\left(q_3,\flat,L\right)$				
q_3	$\left(q_4,\tau,S\right)$	$\left(q_3,\tau,L\right)$	$\left(q_3,\tau,L\right)$					
q_4	$\left(q_4,\tau,R\right)$			$\left(q_f,\flat,S\right)$				

2.2 שקילות מודלים

2.2.1 מבוא

מודל מכונת הטיורינג שהצגנו עונה בבירור על רוב הקריטריונים שהצגנו קודם:

- זהו מודל **סביר** של חישוב - קל לממש אותו, למשל בשפת תכנות מודרנית, או לבנות מכונת טיורינג מלגו, וכדומה.
- זהו מודל **פשוט**: ההגדרה כללה מספר קטן מאוד של רכיבים שכולם מוגדרים היטב.
- זהו מודל **אינטואיטיבי** - או לפחות כך אנחנו מקווים ואם זה לא המצב זו לא אשמתו של טיורינג אלא אשמתנו שלא הצגנו אותו מספיק טוב.

מה שלא ברור כלל הוא שהמודל הזה הוא **כללי** - שכל פונקציה שניתן לחשב באמצעות אלגוריתם, ניתן לחשב באמצעות מכונת טיורינג. כדי להראות זאת אנחנו יכולים "לחזק" את המודל שלנו על ידי הוספת יכולות נוספות, תוך שאנו מראים שהתוצאה היא **שקולה** למודל הקיים - שכל פונקציה שניתנת לחישוב במודל ה"מחוזק" ניתנת לחישוב גם במודל המקורי, אולי במחיר של סיבוך נוסף בבניית המכונה, בזמן הריצה שלה וכדומה.

היעד שלנו הוא להשתכנע שמכונת טיורינג שקולה לכלים שאיתם אנו כותבים אלגוריתמים בחיי היום־יום, דהיינו שפות תכנות. זה יאפשר לנו ליהנות משני העולמות:

- כאשר נרצה לבצע משימה קונקרטית כלשהי בעזרת מכונת טיורינג, נסתפק בלתאר אלגוריתם לא פורמלי עבורה, כזה שברור לנו שאנו יכולים לממש בשפת תכנות, וזה יספיק כדי להשתכנע שאכן **קיימת** מ"ט שמבצעת את המשימה, הגם שהתיאור המלא שלה עשוי להיות מסובך וטרחני.
- כאשר נרצה להוכיח טענה כלשהי על **כל** אלגוריתם, נוכל להסתפק בלהוכיח אותה על **כל** מכונות הטיורינג - ובשל פשטות המודל של מ"ט, ההוכחה הזו תהיה קלה משמעותית יותר.

2.2.2 דוגמא: שקילות למכונה "מהירה"

כדי להמחיש את הרעיון שמאחורי שקילות מודלים, נראה את שקילות המודל הרגיל שלנו למודל שאינו שונה בצורה משמעותית - מכונת טיורינג "מהירה", שהיתרון היחיד שלה הוא בכך שהיא מסוגלת להזיז את הראש הקורא/כותב שלה פעמיים ברצף במקום רק פעם אחת.

פורמלית, המודל החדש זהה למודל הרגיל של מכונת טיורינג למעט פונקציית המעברים, שהיא $\delta:(Q\backslash F)\times\Gamma\to Q\times\Gamma\times\{L,LL,R,RR,S\}$, כאשר אנו מפרשים את RR כהליכת שני צעדים ימינה (כלומר, $j=i+2$) ואת LL כהליכת שני צעדים שמאלה (כלומר, $j=\max\{0,i-2\}$).

כדי להראות שקילות מודלים, יש להראות שתי טענות נפרדות:

- אם M מ"ט רגילה, אז קיימת מ"ט "מהירה" M' כך ש־$f_M=f_{M'}$
- אם M מ"ט "מהירה", אז קיימת מ"ט רגילה M' כך ש־$f_M=f_{M'}$

הטענה הראשונה קלה להוכחה במקרה שלנו, כי מ"ט "מהירה" מהווה הכללה של המודל הקיים - היא מוסיפה לו יכולת, שניתן פשוט להתעלם ממנה. בהינתן מ"ט רגילה M, נגדיר מ"ט "מהירה" M' שזהה לה לגמרי, והיא תחשב את אותה הפונקציה. ההבדל הפורמלי היחיד הוא בטווח של δ', ששונה פורמלית מהטווח של δ (כי הוא $Q\times\Gamma\times\{L,LL,R,RR,S\}$ והטווח של δ הוא $Q\times\Gamma\times\{L,R,S\}$. על פי רוב לא נטרח לציין במפורש הבדלים "פורמליים" כאלו אם הם נפתרים ללא קושי.

על מנת להוכיח את הטענה השניה, תהא $M=(Q,q_0,F,\Gamma,\Sigma,\flat,\delta)$ מ"ט "מהירה" כלשהי. נראה כיצד נבנה מ"ט רגילה המחשבת את אותה הפונקציה.

המכונה שלנו תוגדר בתור $M'=(Q',q_0,F,\Gamma,\Sigma,\flat,\delta')$, דהיינו אנו מבצעים שינויים רק בקבוצת המצבים של M ובפונקציית המעברים שלה. נבצע את השינויים הבאים:

ראשית נגדיר קבוצות מצבים חדשות על בסיס Q הקיימת:

$$Q_R=\left\{q^R\mid q\in Q\right\}$$
$$Q_L=\left\{q^L\mid q\in Q\right\}$$

משמעותם של המצבים בקבוצות אלו הן "הישארי במצב הנוכחי ואל תשני את תוכן הסרט; בצעי צעד אחד בכיוון המתאים".

כעת נגדיר $Q'=Q\cup Q_R\cup Q_L$.

נעבור כעת להגדרת פונקציית המעברים δ' על קלט (p,σ). נפריד בין מספר מקרים:

- אם $p \in Q_R$, כלומר $p = q^R$ עבור $q \in Q$ כלשהו, אז $\delta'(p,\sigma) = (q,\sigma,R)$
- אם $p \in Q_L$, כלומר $p = q^L$ עבור $q \in Q$ כלשהו, אז $\delta'(p,\sigma) = (q,\sigma,L)$
- אם $p \in Q$ נפריד בין המקרה שבו M רוצה לנוע "מהר" והמקרה שבו היא נעה כרגיל. נסמן $\delta(p,\sigma) = (r,\tau,X)$.
  - אם $X \in \{L,R,S\}$ אז $\delta'(p,\sigma) = (r,\tau,X)$.
  - אם $X = RR$ אז $\delta'(p,\sigma) = \left(r^R,\tau,R\right)$.
  - אם $X = LL$ אז $\delta'(p,\sigma) = \left(r^L,\tau,L\right)$.

ניתן להוכיח כי הקונפיגורציה הסופית שאליה תגיע המכונה M' בריצתה על קלט תהיה זהה לקונפיגורציה הסופית של M על אותו הקלט, ואם M אינה עוצרת גם M' לא תעצור, כך שהפונקציות שהן מחשבות זהות. עם זאת, נשים לב כי המכונות עצמן אינן מבצעות את אותו חישוב בדיוק - המכונה M תדלג על קונפיגורציות שבהן M' נמצאת שכן היא מסוגלת לזוז יותר צעדים בבת אחת. זה גם המצב באופן כללי: שקילות מודלים אין פירושה שאנחנו יודעים להמיר מכונה ממודל אחד במכונה ממודל אחר ששקולה לה בכל פרמטר; לעת עתה הפרמטר היחיד שמעניין אותנו הוא מה הפונקציה שהמכונה מחשבת.

2.2.3 דוגמא: שקילות למכונה דו־סרטית

כפי שראינו בדוגמא הקודמת, הוכחת שקילות, גם ברמת הבניה הפורמלית, יכולה להיות דבר מורכב טכנית. לכן נעבור להסברים לא פורמליים, וכדי לפצות על כך נציג מודל שהשיפור בו הוא משמעותי - מכונת טיורינג דו־סרטית. אינטואיטיבית, למכונה כזו יש שני סרטי שלכל אחד מהם ראש קורא/כותב משלו, והמכונה פועלת על שניהם בו זמנית. הסרט הראשון משמש לצורך קריאת הקלט והוצאת הפלט כמו קודם; הסרט השני מיועד רק בתור "שטח עבודה". פורמלית, מודל כזה כולל פונקציית מעברים מורכבת יותר:

$$(Q\backslash F) \times \Gamma^2 \to Q \times \Gamma^2 \times \{L,R,S\}^2$$

ובתחילת ריצתה של המכונה, על הסרט הראשון כתוב הקלט x ואילו הסרט השני כולל רק סימני $\flat$.

מכונה כזו בהחלט מאפשרת לחשב פונקציות מסויימות בצורה **נוחה** יותר. למשל, את הפונקציה $f(x) = xx$ שראינו קודם ניתן לחשב באופן הבא: ראשית, הקלט בסרט הראשון ייסרק משמאל לימין תוך שהוא מועתק לסרט השני; לאחר מכן הראש בסרט השני יוחזר אל תחילת הסרט; לבסוף, הקלט יועתק מהסרט השני אל הסרט הראשון, החל ממקום סיום הקלט המקורי. בצורה זו נחסך למכונה הצורך ללכת שוב ושוב ימינה ושמאלה על גבי הסרט, מה שמוביל לחסכון בזמן ריצה (בהמשך הקורס חסכון זה ודומים לו ייאלץ אותנו להגדיר את המושג של "חישוב יעיל" בצורה רחבה למדי).

ברור שכל מכונת טיורינג רגילה יכולה להיחשב למקרה פרטי של מכונה דו־סרטית (שפשוט לא משתמשת בסרט הנוסף שלה) כך שכיוון זה של הוכחת שקילות המודלים הוא קל. בכיוון השני נשאלת השאלה - כיצד מכונה חד־סרטית יכולה להתמודד עם הצורך לטפל בשני סרטים במקביל? יש כאן כמה נקודות שדורשות התייחסות:

- **ייצוג שני הסרטים**: דרך פשוטה לייצג את שני הסרטים על סרט אחד היא לכתוב ראשית את תוכן הסרט הראשון, להוסיף סימן מפריד מיוחד (למשל \$, בהנחה שאינו שייך לא"ב העבודה של המכונה המקורית), ואז לכתוב את תוכן הסרט השני. הקושי בגישה זו נובע מכך שבכל פעם שבה המכונה חורגת בסרט הראשון מהאיזור שנצפה עד כה, צריך יהיה להזיז ימינה את כל תוכן הסרט השני כדי "לפנות מקום" לסרט הראשון. זו אינה בעיה אמיתית שכן כבר ראינו כיצד ניתן להזיז את תוכן הסרט צעד אחד ימינה.

- **זיהוי מיקום הראשים הקוראים**: בכל צעד שלה, מכונת הטיורינג שלנו צריכה לראות מה שני הראשים הקוראים/כותבים של המכונה המקורית "רואים". לשם כך נוסיף סימון מיוחד לכל תו בקלט שמשמעותו "הראש הקורא נמצא כאן". למשל במקום σ נכתוב σ'.

- **ביצוע צעד חישוב**: כל צעד חישוב של המכונה המקורית יתחיל בכך שהראש הקורא של המכונה החדשה נמצא על האיזור שמתאים לסרט הראשון, במקום של הראש הראשון. בשלב זה המכונה תקרא ותזכור את התו שבמקום זה, תנוע ימינה עד למיקום הראש הקורא על הסרט השני, תזכור מה הפעולה שהמכונה המקורית מבצעת, תבצע את הפעולה המתאימה על הסרט השני, תחזור אל מיקום הראש של הסרט הראשון ותבצע את הפעולה שהוא אמור לבצע. בצורה זו אם החישוב מסתיים, הראש של המכונה נמצא במקום הנכון - בדיוק במיקום הראש הראשון של המכונה המקורית, ולכן המכונה תחזיר את אותו הפלט.

2.2.4 מודל ה־RAM

ראינו כיצד ניתן "לשפר" מכונת טיורינג עד לקבלת מודל נוח יותר לעבודה, אולם מכונת טיורינג היא עדיין מודל שונה למדי מזה שאנחנו עוסקים בו ביום יום כאשר אנו כותבים קוד. הבדל מובהק אחד הוא שאנחנו כותבים קוד למכונה בעלת RAM ־ זכרון "גישה אקראית" שבו אנחנו יכולים לגשת לתאים ספציפיים בעזרת **הכתובת** שלהם, מבלי שנזדקק בפועל לטיול על גבי סרט כדי להגיע אליהם.

כאשר אנו כותבים קוד בשפה עילית, הוא עובר תהליך של קומפילציה לשפת אסמבלר (או שהוא מורץ בידי מפרש שבתורו עבר תהליך קומפילציה כזה). כלומר, מספיק לנו להשתכנע בכך שמכונת טיורינג יכולה לסמלץ את הפעולות שמבוצעות בשפת אסמבלר. שפות אסמבלר מיועדות למעבדים רבים ושונים, אבל בבסיסן יש להן מרכיבים זהים:

- הזיכרון הוא זיכרון גישה אקראית.
- התוכנית אותה מריצים שמורה כחלק מהזיכרון.
- יש למעבד רכיבי זכרון המכונים **רגיסטרים** שמיועדים להכיל כמות קטנה של מידע.
- אחד מהרגיסטרים של המעבד מצביע על המקום שמתבצע בתוכנית.
- המעבד מבצע פעולות של:
  - קריאה מהזכרון האקראי לתוך רגיסטר.
  - כתיבה מרגיסטר לתוך הזכרון האקראי.
  - ביצוע פעולה מתמטית־לוגית כלשהי על רגיסטרים.

קריאה וכתיבה מהזכרון האקראי מתבצעות על ידי כך שרגיסטר אחר שומר את הכתובת של תא הזיכרון שאליו רוצים לקרוא ולכתוב.

דוגמאות לפעולות מתמטיות־לוגיות: ביצוע פעולת חיבור של שני רגיסטרים; ביצוע NOT על תוכן רגיסטר; השוואת תוכן רגיסטר לאפס ושינוי ערכו של רגיסטר מיקום התוכנית בהתאם לכך, וכדומה.

כיצד ניתן לסמלץ מכונת RAM שכזו באמצעות מכונת טיורינג? יש דרכים רבות, אבל מכיוון שאיננו מתעניינים ב**יעילות** של הסימולציה אלא רק בהוכחת היתכנות קיומה, נבחר דרך פשוטה אך "בזבזנית" במיוחד.

אם במכונת ה־RAM ישנם n רגיסטרים, נבנה מ"ט בעלת $n+1$ סרטים. לכל רגיסטר יש סרט משלו, והסרט האחרון מוקדש לתוכן הזכרון.

ביצוע פעולות בין שני רגיסטרים אינו עניין מסובך; קל להראות במפורש מ"ט בעלות שני סרטים שיודעות לחבר, לחסר, לכפול וכו', בהתבסס על האלגוריתמים לפעולות החשבון ("חיבור ארוך" וכדומה). האתגר נעוץ בשתי התכונות המרכזיות של מודל ה־RAM:

- הזכרון האינסופי שממוען לפי כתובות.
- העובדה שה"תוכנית" של מודל ה־RAM כתובה כחלק מהזכרון בתחילת הריצה, ובמ"ט לא כתוב כלום על הסרט בתחילת הריצה.

את בעיית הזכרון האינסופי נפתור, כאמור, בדרך בזבזנית. סרט הזיכרון שלנו יורכב מסדרת תאים מהצורה

$$\#A_1\$C_1\#A_2\$C_2\ldots\#A_m\$C_m\#$$

כאשר $A_1,A_2,\ldots$ הם מספרים שבאים לציין **כתובת** של תא, ו־$C_1,C_2,\ldots$ באים לתאר את **תוכן** התא. כלומר, המופע של $\#A_i\$C_i$ על הסרט בא לומר "בתא שכתובתו היא A_i מצוי המידע C_i".

הדרך **לקרוא** מתוך התא שהכתובת שלו היא D היא לעבור על כל סרט הזיכרון ולחפש מופע של $\#D\$$. אם נמצא כזה, מועתקת אל הרגיסטר המתאים המחרוזת שמימין ל־$, עד למופע הבא של #; אם לא נמצא מופע כזה, הערך המוחזר הוא ε.

הדרך **לכתוב** C לתוך התא שכתובתו היא D היא לעבור על כל סרט הזיכרון ולחפש מופע של $\#D\$$. אם לא נמצא כזה, מוסיפים $\#D\$C$ אל סוף הסרט; אם נמצא כזה, אפשר לדחוף את C במקום התוכן שנמצא כרגע בתא ולהזיז את כל יתר הסרט בהתאם; אפשר גם "לבטל" את התא על ידי החלפת $\#D\$$ ב־$\#DX$ כאשר X הוא סימן אחר שאינו בשימוש, ואז להוסיף $\#D\$C$ לסוף הסרט.

2.2.5 התזה של צ'רץ' וטיורינג

ראינו את המודל של מכונת טיורינג, וראינו גם מספר הרחבות אליו, כולל אחת שמזכירה במעורפל את אופן פעולתו של מחשב אמיתי. כל ההרחבות הללו היו שקולות בכוחן החישובי, במובן זה שכל מה שניתן לחישוב במודל אחד, ניתן לחישוב גם במודל האחר. השקילות הזו לא מביאה בחשבון שיקולים נוספים כמו זמן הריצה (שאליו נתייחס בחלקו השני של הקורס), כמות המקום שתופס החישוב, כמה מסובך התיאור של ה"תוכנית" שמריץ המודל, צריכת האנרגיה של המכונה וכדומה. כל אלו הם בפני עצמם שיקולים חשובים אך מצריכים דיון מורכב יותר מזה שביצענו עד כה.

המקור ההיסטורי של מכונות טיורינג היה בנסיון למדל "אלגוריתם" בצורה מתמטית מדויקת דיה כדי שניתן יהיה לטעון בביטחון טענה מסוג "לא קיים אלגוריתם אשר פותר את הבעיה הבאה..." - טענה שאכן לא מצריכה התייחסות לשיקולים הנוספים לעיל. למודל של טיורינג קדם מודל אחר, תחשיב הלמבדא של צ'רץ', שהשיג את אותה המטרה; ולשניהם קדם מודל נוסף, הפונקציות הרקורסיביות של קורט גדל. גדל השתמש בפונקציות רקורסיביות במאמר שבו הוכיח את משפטי אי השלמות המפורסמים שלו, אך רק לאחר הרחבה מאוחרת יותר שלהן התקבל מודל כללי ששקול למודלים של צ'רץ' וטיורינג.

האופן שבו התפתחו מספר מודלים שונים בצורה מהותית באופיים ותיאורם אך שקולים בכוחם החישובי הוביל לתחושה ש"זה לא במקרה": שכל מודל שיהיה חזק יותר ממכונת טיורינג, ייאלץ לצורך כך להפוך לבלתי סביר למימוש בפועל. למשל, אם נרשה למכונת טיורינג לחזור בזמן או להיכנס לחור שחור, זה עשוי להרחיב את יכולתה החישובית, אך בהינתן חוקי הפיזיקה הידועים לנו נראה בלתי סביר שנוכל לממש מכונה שכזו בפועל.

מכאן הגיעה ההשערה המכונה **התזה של צ'רץ' וטיורינג**: כל המודלים החישוביים הסבירים והכלליים שקולים זה לזה. כמובן, יש להסביר את משמעות המילים "סביר" ו"כללי" בהקשר זה; "כללי" פירושו שהמודל אינו מוגבל מדי (למשל, המודל של **אוטומט סופי דטרמיניסטי** אינו כללי; הוא שקול למכונת טיורינג שפועלת בזיכרון עבודה קבוע). "סביר" פירושו הטענה המעורפלת שבה עסקנו קודם, על היכולת לממש את המודל בפועל. מכיוון שהתזה אינה משפט אלא השערה, או "הנחת עבודה", אין צורך בניסוח מדויק יותר שלה.

קרוב למאה שנים חלפו מאז הועלתה התזה ועד כה טרם תואר מודל חישובי שמפר אותה. עם זאת, כדאי להעיר על מודל אחר שהוא בעל פוטנציאל להפר גרסה מורחבת של התזה, שהוצעה שנים רבות אחרי צ'רץ' וטיורינג.

בהמשך הקורס נעסוק במכונות שמוגבלות מבחינת זמן הריצה שלהן, וניתן הגדרה מסוימת למהו זמן ריצה "יעיל". התזה המורחבת של צ'רץ' וטיורינג מניחה שכל המודלים החישוביים הסבירים, כלליים ויעילים שקולים בכוחם החישובי, כלומר שאם בעיה כלשהי ניתנת לפתרון יעיל באחד המודלים הללו, היא ניתנת לפתרון יעיל בכל אחד אחר מסוג זה. בשנים האחרונות טענה זו עומדת למבחן אל מול המודל של **חישוב קוונטי**. ההשערה היא שחישוב קוונטי יעיל מסוגל לפתור בעיות שלא ניתנות לפתרון יעיל במודל של מכונת טיורינג - דוגמא מפורשת אחת היא בעיית **הפירוק לגורמים** של מספרים, שניתנת לפתרון יעיל במחשב קוונטי. עם זאת, כיום אין לנו **הוכחה** לכך שפירוק לגורמים לא ניתן לפתרון יעיל במכונת טיורינג, וגם השאלה האם ניתן לממש מחשבים קוונטיים במציאות כך שיהיו מסוגלים לפתור בפועל את בעיית הפירוק לגורמים בצורה יעילה יותר ממחשבים רגילים עדיין לא זכתה למענה משביע רצון (קיימים בפועל מחשבים קוונטיים, אך יכולת החישוב שלהם עודנה מוגבלת מאוד עקב רעשים, והשאלה ההנדסית עד כמה ניתן לקדם את התחום עודנה פתוחה).

2.3 מכונת טיורינג אוניברסלית

2.3.1 מבוא

מטרתו של טיורינג בהמצאת המודל שלו הייתה להוכיח את אי־הכריעות של בעיות אלגוריתמיות מסויימות. ההוכחה שלו שאבה השראה מהוכחת משפטי אי השלמות של קורט גדל ב־1931, שעסקו במגבלות של מערכות הוכחה מסוימות בלוגיקה מתמטית. מבלי להיכנס לפרטי ההוכחה של גדל, רעיון מרכזי ומבריק שלו היה לקחת מערכת הוכחה שמיועדת לדיבור על תכונותיהם של מספרים טבעיים עם פעולות החיבור והכפל, ולגרום לה לדבר **על עצמה**, על ידי כך שטענות והוכחות במסגרת מערכת ההוכחה עצמה מקודדות על ידי מספרים טבעיים. כך השאלה "האם קיימת הוכחה לטענה X מתוך אוסף האקסיומות $A_1,\ldots,A_k$" ניתנת לתרגום לשאלה "האם קיימת הוכחה לטענה שמקודדת באמצעות המספר הטבעי n_X מתוך אוסף האקסיומות שמקודדות באמצעות המספרים הטבעיים $n_{A_1},\ldots,n_{A_k}$".

בדיקת הוכחות אוטומטית היא בימינו עניין לא קשה לביצוע; כל צעד בהוכחה הוא **כלל היסק** שלוקח מספר מחרוזות ומסיק מתוכן מחרוזת חדשה על בדיקה של טקסט המחרוזות עצמן, בלי צורך בהבנה של המשמעות שהן מייצגות. למשל, כלל ההיסק מודוס פוננס מקבל את המחרוזות $A\to B$ ו־A ומסיק מהן את B - קל לכתוב קוד שמבצע את הבדיקה המתאימה ומחזיר את הפלט המתאים. מכיוון שמחרוזות מקודדות במחשב מודרני באמצעות מספרים (למשל, בקידוד ASCII או UTF) בפועל מה שמתרחש מאחורי הקלעים הוא בדיקה האם ממספרים טבעיים נובע מספר טבעי אחר על פי כללי היסק מסויימים. במאמר שלו, קורט גדל ביצע את העבודה הטכנית של כתיבת "קוד שמבצע את הבדיקה המתאימה" גם מבלי שיהיה לו מושג של מחשב או שפת תכנות להסתמך עליו.

אלן טיורינג לקח את הרעיונות של גדל לשלב הבא: הוא רצה לאפשר למכונות טיורינג לדבר **על מכונות טיורינג**. מכיוון שמכונות טיורינג מקבלות כקלט מחרוזת, השלב הקריטי היה להראות שניתן לקודד מכונת טיורינג M בתור מחרוזת $\langle M\rangle$,

בצורה כזו שניתן יהיה להשתמש במחרוזות כדי לבצע את פעולת המכונה. מכיוון שמה שמכונת טיורינג עושה הוא לרוץ על קלטים, הרעיון של טיורינג היה לבנות מכונת טיורינג U, שנקראת **מכונת טיורינג אוניברסלית**, שמקבלת קלט שהוא **זוג** $(\langle M\rangle,\langle x\rangle)$ שכולל קידוד של מכונת טיורינג M כלשהי, וקידוד של קלט x כלשהו עבור M, ומה ש־U יודעת לעשות הוא לבצע סימולציה של ריצת M על x. כלומר, ליצור סדרתית ייצוג של הקונפיגורציות בריצת M על x, ואם M עוצרת ־ לזהות זאת, ולדעת מה הפלט ש־M מחזירה על x.

כמובן, ניתן להשתמש ב־U האוניברסלית גם בצורה חכמה יותר מאשר "סתם" להריץ מכונה על קלט; ניתן להריץ מכונה על שני קלטים **במקביל**, או אפילו להריץ אינסוף מכונות שונות על אינסוף קלטים במקביל, צעד אחד בכל פעם לכל אחת מהמכונות. היכולת החדשה הזו של מכונות טיורינג ־ היכולת **להריץ מכונות טיורינג** היא המפתח לכל מה שנעשה בהמשך, ובפרט להוכחה שיש בעיות אלגוריתמיות שאינן כריעות.

2.3.2 קידוד

במכונה האוניברסלית שנציג נרצה לשמור על פשטות הייצוג ככל שניתן, וזאת במחיר גבוה ביעילות הפעולה של המכונה ־ מה שלא רלוונטי לנו כלל בהקשר הנוכחי. לשם כך נקבע שהמכונה האוניברסלית U שלנו תהיה בעלת א"ב העבודה $\Gamma=\{0,1,\flat\}$ וא"ב הפלט $\Sigma=\{0,1\}$ בלבד.

כאשר U מקבלת קלט M,x אנחנו ניצבים בפני הבעיות הבאות:

- מספר המצבים Q של M אינו חסום; כלומר, U צריכה לדעת להתמודד עם M ולא משנה כמה גדול מספר המצבים של M.
- גודל א"ב העבודה של M אינו חסום, בדומה לגודל Q.
- הקלט והפלט של M צריכים להינתן באמצעות א"ב הקלט והעבודה של M, לא של U.

כדי לפתור את הבעיות הללו אנחנו משתמשים בקידוד ־ ייצוג הן של המכונה M והן של הקלט x באמצעות מחרוזות של תווים מתוך $\{0,1\}^*$. את הקידוד של M נסמן $\langle M\rangle$ ואת הקידוד של x נסמן $\langle x\rangle$.

ראשית נסביר כיצד לקודד מחרוזת $x\in\Gamma^*$. באופן כללי, אברי Γ ניתנים להצגה בתור $\Gamma=\{a_1,a_2,\ldots,a_n\}$. כדי לפשט עוד יותר את הסימונים, נוכל להניח שהאיברים מיוצגים באמצעות האינדקס שלהם, כלומר $\Gamma=\{1,2,\ldots,n\}$. לצורך פשטות נבחר את המספור בדרך כזו שמתקיים $\Sigma=\{1,2,\ldots,m\}$ כך ש־$m<n$, כלומר במספור אברי Γ רשומות קודם האותיות ששייכות גם ל־Σ.

נזכיר מהו **ייצוג אונרי** של מספר טבעי: זו פשוט מחרוזת של 1־ים שאורכה כגודל המספר. כך למשל המספר "שלוש" מיוצג על ידי 111 והמספר "שמונה" על ידי 11111111. זוהי שיטה בזבזנית מאוד לייצוג, מבחינת מספר התווים שנדרשים לייצג כל מספר, אך היא תספיק לנו. הרעיון הוא שנקודד מחרוזת על ידי **ייצוג אונרי של האינדקס של התווים שבה** כאשר 0 משמש לנו בתור תו מפריד בין אותיות שונות.

כלומר, ראשית נגדיר $\langle i\rangle=1^i$ לכל $i\in\Gamma$, וכעת עבור $x=x_1\cdots x_k\in\Gamma^*$ נגדיר

$$\langle x\rangle=\langle x_1\rangle 0\langle x_2\rangle 0\cdots 0\langle x_k\rangle 0$$

כך למשל, אם $\Gamma=\{a,b,c\}$ אז המחרוזת $acab$ תיוצג על ידי 10111010110.

נעבור כעת לקידוד של המכונה M.

כזכור, מ"ט כוללת את המרכיבים הבאים: $M=(Q,q_0,F,\Gamma,\Sigma,\flat,\delta)$. את רובם ניתן לקודד בפשטות בזכות היכולת שלנו להניח כמה הנחות מקילות.

- כמו קודם, אנו מניחים כי $\Gamma=\{1,2,\ldots,n\}$ כך ש־$\Sigma=\{1,2,\ldots,m\}$ כוללת את חלק מהתווים בתחילת Γ ומספיק לנו להתייחס לתווים באמצעות האינדקס שלהם בלבד.
- נניח כי $\flat=n$, כלומר התו האחרון ב־Γ (שבודאות אינו שייך ל־Σ) הוא זה שמייצג את $\flat$.
- עבור Q נפשט את סימוני האיברים בצורה דומה: נניח כי $Q=\{1,2,3,\ldots,|Q|\}$ כלומר כל מצבי Q הם מספרים טבעיים החל מ־1. נניח ש־$q_0=1$, כלומר המצב ההתחלתי הוא המצב הראשון ב־Q.
- נניח כי $F=\{2,3\}$. הנחה זו דורשת הסבר נוסף. אם ב־M היו פחות מצבים סופיים, אפשר להוסיף עוד מצבים כרצוננו בלי לשנות את התנהגות המכונה. אם היו בה יותר מצבים, אפשר "למזג" אותם על ידי שינוי פונקציית המעברים (במקום להעביר למצב q_f מסוים, להעביר תמיד אל 2, מה ש"ממזג" את q_f עם 2). מכיוון שריצת המכונה נעצרת אחרי

כניסה למצב סופי, המיזוג הזה לא משפיע על התנהלות המכונה. נשאלת אם כן השאלה מדוע אנו זקוקים **לשני** מצבים סופיים ולא לאחד; התשובה היא שבהמשך נראה מכונות ("מכונות להכרעת שפות") שיש בהן בדיוק שני מצבים סופיים ויש חשיבות לשאלה לאיזה מצב סופי המכונה מגיעה.

- פונקציית המעברים δ כזכור מקיימת $\delta(q,\sigma)=(p,\tau,X)$, כלומר מקבלת שני קלטים ומחזירה שלושה פלטים. אפשר לחשוב על δ בתור קבוצה של חמישיות (q,σ,p,τ,X) ולקודד כל חמישייה בנפרד. כבר ראינו שכל מצב ואות הם מספרים, ואפשר להניח שגם $\{S,L,R\}$ מיוצגת באמצעות מספרים (למשל $S=1,L=2,R=3$) ולכן

$$\langle\delta(q,\sigma)\rangle=\langle(q,\sigma,p,\tau,X)\rangle=1^{q}01^{\sigma}01^{p}01^{\tau}01^{X}$$

בהינתן כל אלו, ניתן לקודד את M כולה באופן הבא:

$$\langle M\rangle=1^{|Q|}01^{|\Gamma|}01^{|\Sigma|}00\left\langle\delta(1,1)\right\rangle0\left\langle\delta(1,2)\right\rangle0\left\langle\delta(1,3)\right\rangle0\cdots\left\langle\delta(|Q|,|\Gamma|)\right\rangle00$$

כאשר ה־00 לפני ואחרי כתיבת ה־δ מאפשר לנו לזהות את תחילת וסוף האיזורים שבהם פונקציית המעברים מופיעה.

2.3.3 סימולציה

נסביר כעת כיצד לבנות מ"ט אוניברסלית U שמקבלת $\langle M\rangle,\langle x\rangle$, מריצה את M על x ומחזירה את $\langle f_M(x)\rangle$ אם M עוצרת על x. במאמרו, טיורינג נתן תיאור מלא של מכונה כזו, אך אנו נסתפק בהסבר לא פורמלי (זו הצורה שבה אנו חומקים מהליבה הטכנית של הקורס, שהיא מה שמאפשר ליתר הקורס להיות לא פורמלי יחסית).

כבר ראינו איך מ"ט דו־סרטית היא שקולה למ"ט חד־סרטית; באותו האופן ניתן להוסיף עוד מספר סופי כלשהו של סרטים. כדי להקל על עצמנו נניח של־U יש 4 סרטים.

U תפעל בצורה הבאה: ראשית היא תוודא את **תקינות הקלט** - כלומר, שהקלט שלה, שאמור להיות $(\langle M\rangle,\langle x\rangle)$ אכן נראה כמו קידוד של מ"ט ושל מחרוזת. לשם כך היא תוודא שתחילת הקלט שלה הוא מהצורה הבאה:

- מספר גדול מ־0 של 1־ים (זה $|Q|$) ואז 0. את רצף ה־1־ים הזה המכונה תעתיק לסרט מס' 2 שלה, שנכנה "סרט המצבים".
- מספר גדול מ־0 של 1־ים (זה $|\Gamma|$) ואז 0. את רצף ה־1־ים הזה המכונה תעתיק לסרט מס' 3 שלה, שנכנה "סרט האלפבית".
- מספר גדול מ־0 של 1־ים (זה $|\Sigma|$) ואז 0. תוך קריאת ה־1־ים הללו נשווה את מספרם למספר ה־1־ים בסרט האלפבית ונוודא שהוא קטן יותר (כלומר, אחרי שאנחנו מגיעים אל ה־0 בסיום עדיין לא הגענו אל קצה ה־1־ים בסרט האלפבית).
- 00 ואחריו כפולה של 5 של קטעי 1־ים מופרדים באפסים בודדים, ואז 00 נוסף. את אוסף החמישיות הזה המכונה תעתיק לסרט מס' 4 שלה, שנכנה "סרט פונקציית המעברים".
- את כל מה שיש אחרי ה־00 (שיכול להיות מחרוזת ריקה) מעתיקים לתחילת הסרט ומוחקים את כל מה שהיה משמאל אליו.

אם בשלב כלשהו בוידוא ההתחלתי הזה דברים אינם מתנהלים כפי שהם אמורים, כלומר ה"קידוד" של $\langle M\rangle$ אינו קידוד חוקי, המוסכמה שלנו היא ש־U תרוץ לנצח (למשל, תיכנס למצב שבו היא תמיד מבצעת צעד ימינה ונשארת באותו מצב). אינטואיטיבית, כל קידוד שהוא ג'יבריש מקודד לנו מכונה $\langle M_{stam}\rangle$ שלא עוצרת על אף קלט.

בדיקה נוספת שיש לבצע בשלב זה היא שפונקציית המעברים δ היא חוקית - כלומר, שלכל $1\le i\le|Q|$ ו־$1\le j\le|\Gamma|$ מופיע הקלט (i,j) בדיוק פעם אחת, ולפי הסדר הלקסיקוגרפי. ניתן לבצע את הבדיקה הזו בקלות עם שני סרטי עזר שאחד כולל את i והשני את j. במקרה והבדיקה נכשלה, אנו מניחים כמו קודם שהמכונה היא $\langle M_{stam}\rangle$.

כדי לסמלץ חישוב, המכונה האוניברסלית צריכה בכל רגע נתון לשמור על הסרט את ה**קונפיגורציה** של M. את הקונפיגורציה $C=(\alpha,q,i)$ ניתן לייצג בתור המחרוזת הבאה, שהיא פשוט α עצמה שבה האיבר ה־i מוחלף בזוג (α_i,q):
$\alpha_1\alpha_2\cdots(\alpha_i,q)\cdots\alpha_t$.
קידוד של הקונפיגורציה הזו יהיה

$$\begin{aligned}\langle \alpha_1\alpha_2\cdots(\alpha_i,q)\cdots\alpha_t\rangle &= \langle\alpha_1\rangle\,0\,\langle\alpha_2\rangle\,0\cdots\langle(\alpha_i,q)\rangle\cdots\langle\alpha_t\rangle\,0\\ &= 1^{\alpha_1}01^{\alpha_2}0\cdots01^{\alpha_i}001^{q}00\cdots1^{\alpha_t}0\end{aligned}$$

כלומר, ניתן לזהות את מיקום q על ידי 0־ים כפולים משני הצדדים.

בתחילת החישוב, מה שכתוב על הסרט הוא $\langle x\rangle$ (את $\langle M\rangle$ כבר מחקנו). המכונה מוסיפה לתו הראשון ב־x את $001^{q_0}00$, כלומר את 00100. אם x הוא המחרוזת הריקה, המכונה תיצור את $\langle(\flat,q_0)\rangle$. שימו לב שכדי ליצור את $\langle\flat\rangle=1^{|\Gamma|}$ צריך לדעת את $|\Gamma|$ - זה מתבצע על ידי העתקת המחרוזת שבסרט האלפבית.

כעת, בכל צעד חישוב, U תפעל בצורה הבאה:

- תעביר את הראש בסרט הראשון אל הישר מימין ל־00 השמאלי יותר, כלומר תחילת q, ואת הראש בסרט פונקציית המעברים אל תחילת הסרט.
- תעבור סדרתית על האברים $\langle\delta\left(a,b\right)\rangle$ בסרט פונקציית המעברים ותשווה כל איבר כזה עם הקונפיגורציה הנוכחית. כלומר:
  - תבדוק ש־$1^q=1^a$ על ידי מעבר עם שני הראשים (זה של הקונפיגורציה וזה של פונקציית המעברים) בו זמנית צעד־צעד ובדיקה אם הגיעו ביחד אל 0.
  - במידה שהבדיקה הקודמת עבדה, תעבור בסרט הקונפיגורציה אל α_i על ידי מעבר שמאלה על פני ה־00 ועד ל־0 הקודם, ואז תשווה את α_i אל 1^b.
  - במידה ואחת משתי הבדיקות הקודמות נכשלו, המכונה תעבור אל תחילת החמישייה הבאה בסרט פונקציית המעברים (מובטח לנו שאחת מהבדיקות תצליח, שכן קידוד פונקציית המעברים הוא חוקי).
- כאשר שתי הבדיקות הצליחו, כלומר נמצאה חמישייה (q,σ,p,τ,X) המכונה תשנה את הקונפיגורציה הנוכחית בהתאם:
  - את α_i המכונה תמחק, ותעתיק את 1^τ במקומו.
  - אם $X=S$, המכונה תחליף את 1^q ב־1^p. כך גם תפעל המכונה אם $X=L$ אבל $i=0$ (כלומר, אנחנו בקצה השמאלי של הקלט).
  - אם $X=L$, המכונה תמחק את 001^q00, תלך אל משמאל ל־0 של α_i כך שהיא בקצה הימני של התא של α_{i-1}, ותכתוב שם 001^p0 (ה־0 שהיא עקפה יצטרף אל ה־0 הימני).
  - אם $X=R$ המכונה תפעיל בדומה למקרה של $X=L$ אלא אם כן α_i היה בקצה הימני של הקלט (מה שניתן לזהות על ידי כך ש־U תיתקל מעבר לקצה זה בסימן ה־$\flat$ של הא"ב שלה. במקרה זה, U תכתוב מעבר ל־0 הימני ביותר את המחרוזת $1^{|\Gamma|}001^p00$ (שמייצגת את $\langle(\flat,p)\rangle$).
- אם $p\in F$, המכונה תעביר את עצמה אל הקצה השמאלי של התא 1^t001^p00 שהיא נמצאת בו ותעצור (כך שהפלט שלה יהיה הקידוד של המחרוזת עד ולא כולל התו הנוכחי).

את המסקנה מכל הדיון הזה ניתן לרכז לכדי משפט מחץ אחד:

משפט 7.2 נגדיר את "הפונקציה האוניברסלית" U בתור הפונקציה שעל קלט $U\left(\langle M\rangle,\langle x\rangle\right)$, אם המ"ט M עוצרת על x אז $U\left(\langle M\rangle,\langle x\rangle\right)=\langle f_M\left(x\right)\rangle$ ואחרת $U\left(\langle M\rangle,\langle x\rangle\right)$ אינה מוגדרת. אז U ניתנת לחישוב.

המשמעות הפרקטית מכאן ואילך של קיום מ"ט אוניברסלית היא שנרשה לעצמנו, בעת בניית מ"ט כלשהי, להגיד "המכונה שלנו תפרש את הקלט שלה בתור מ"ט M" ו"המכונה שלנו תריץ את M בצורה כך וכך" וכדומה. יכולת זו מתקבלת מכך שאחד מרכיבי המ"ט שנבנה יכלול את המ"ט האוניברסלית U שהסברנו כיצד לבנות.

2.4 משפט הרקורסיה של קלייני

נסיים חלק זה בנושא מתקדם יחסית, שלא נזדקק לו בהמשך אם כי הוא יכול לפשט חלק מהדברים שנעשה. השאלה הבסיסית שנרצה לענות עליה היא "האם בזמן שבונים מ"ט M ניתן להניח כי הקידוד $\langle M\rangle$ יהיה ידוע ל־M והיא תוכל להשתמש בו?" והתשובה היא "כן".

כדי לקבל אינטואיציה לכך שזו אינה שאלה טריוויאלית, נזכיר שאלה דומה הקשורה לה בקשר הדוק ־ האם קיימת תוכנית מחשב שמדפיסה את קוד המקור של עצמה, וזאת מבלי לנקוט בתעלולים כמו פתיחת קובץ שבו כתוב הקוד? התשובה לכך גם כן חיובית, וקיימות אינספור דרכים לכתוב תוכניות כאלו, שזכו לשם quine על שם הפילוסוף בשם זה. עם זאת, כל נסיון נאיבי לכתוב תוכנית כזו יוביל מהר מאוד לבעיה שתסייע להבנה האינטואיטיבית של הקושי כאן: אם התוכנית סתם תנסה לכלול פקודת print ואחריה מחרוזת הכוללת את תוכן התוכנית, תיווצר הבעיה לפיה חלק מתוכן התוכנית הוא אותה מחרוזת עצמה שאותה מנסים להדפיס, ונסיון לכלול אותה בפנים יאריך אותה עוד ועוד, עד אין קץ. צריך לנקוט בחיסכון בדרך כלשהי.

ראשית נפתור את בעיית ה־quine בהקשר של מכונות טיורינג. כלומר, נבנה מ"ט M שעל הקלט ε מחזירה את הפלט $\langle M\rangle$. לצורך כך, נתחיל בצורה צנועה אף יותר: נבנה מכונות טיורינג A,B כך שאם A מורצת על הקלט הריק ואז B מורצת על הפלט של A, התוצאה היא $\langle A\rangle\langle B\rangle$, כלומר $f_B\left(f_A\left(\varepsilon\right)\right)=\langle A\rangle\langle B\rangle$.

המכונה A עומדת להיות פשוטה מאוד: זו מכונה שעל הקלט ε כותבת את הפלט $\langle B\rangle$ ועוצרת. אלא שההגדרה הזו בעייתית כי טרם הגדרנו את B; נגדיר את B בזהירות מבלי להתייחס כלל ל־A כדי שלא ליצור הגדרה מעגלית.

אינטואיטיבית, B היא מכונה שעל קלט w כותבת על ידו קידוד $\langle M_w\rangle$ של מ"ט M_w שהיא מכונה שעל הקלט ε מוציאה כפלט w. כיצד B עושה זאת?

בהינתן מילה $w=\sigma_1\sigma_2\cdots\sigma_n$, מכונה שעל ε כותבת את w יכולה להיות מכונה שמצביה הם $Q=\{1,2,\ldots,n,n+1\}$ כך ש־$F=\{n+1\}$ והמעברים שלה הם כולם $\delta\left(q_i,\flat\right)=\left(q_{i+1},\sigma_i,R\right)$ (המעברים על תווים שאינם $\flat$ יכולים להיות משהו שרירותי שכן בכל מקרה לא נשתמש בהם). קל לכתוב את הקידוד של מכונה זו ־ הדבר היחיד שתלוי ב־w עצמה הוא ה־σ_i שמופיעים בחלק של פונקציית המעברים. נסמן בתור q את הפונקציה שעל קלט w מחזירה את המכונה M_w המתאימה לתיאור לעיל ־ פורמלית $q\left(w\right)=\langle M_w\rangle$. כאמור, q ניתנת לחישוב בקלות יחסית.

כעת נגדיר את B כך: על קלט w היא מחשבת את $q\left(w\right)$ ומוציאה כפלט את $q\left(w\right)w$. שימו לב שבהגדרה זו לא הסתמכנו על A.

כעת נגדיר את המכונה A על ידי $A=q\left(\langle B\rangle\right)$. כלומר, A אינה "סתם" מכונה שעל ε מוציאה $\langle B\rangle$ כפלט, אלא אותה מכונה ממש ש־B תייצר אם תופעל על הקלט $\langle B\rangle$.

כעת, פעולתה של B על הקלט $\langle B\rangle$ הוא כדלהלן: היא כותבת על הסרט את $q\left(w\right)w$, כלומר במקרה שלנו את

$$q\left(\langle B\rangle\right)\langle B\rangle=\langle A\rangle\langle B\rangle$$

כפי שרצינו.

לרוע המזל, המכונה שקודם מפעילה את A ואז מפעילה את B על התוצאה **אינה** מקודדת באמצעות $\langle A\rangle\langle B\rangle$ ־ קידוד של מכונה אינו מתאים לזוג קידודי מכונות זה לצד זה. אלא ש**ניתן לחשב** את הקידוד של מכונה שכזו מתוך הקידודים $\langle A\rangle,\langle B\rangle$. אם כן, נוכל לשפר את המכונה B שלנו בצורה הבאה:

- על קלט w, B מחשבת את $q\left(w\right)$.
- B מפרשת את w ואת $q\left(w\right)$ בתור קידודים של מכונות טיורינג.
- B בונה קידוד של מ"ט שמפעילה קודם את המכונה $q\left(w\right)$ ואז מפעילה את המכונה w על התוצאה. את הקידוד הזה B כותבת כפלט.

נסמן ב־$\langle AB\rangle$ את הקידוד הנוצר מ"שילוב" כזה של A,B. נשים לב לכך שהמכונה AB שמקודדת באמצעות $\langle AB\rangle$ היא מכונה שמפעילה את A קודם ואז את B על הפלט, ולכן על הקלט ε המכונה AB תוציא את הפלט $\langle AB\rangle$, כמבוקש.

בזאת הוכחנו את המשפט הבא:

משפט 8.2 קיימת מ"ט M כך ש־$f_M\left(\varepsilon\right)=\langle M\rangle$

נרצה להכליל את מה שעשינו עד כה עבור טענה חזקה יותר: שכאשר אנו בונים מכונת טיורינג, אנחנו יכולים להניח שהמכונה מכירה את הקידוד של עצמה ויכולה להשתמש בו באופן חופשי. הניסוח הפורמלי של הטענה הזו נתון במשפט הבא:

משפט 9.2 תהא M מ"ט המתייחסת לקלט שלה בתור הזוג $\left(x,y\right)$. אז קיימת מ"ט M' כך ש־$f_{M'}\left(x\right)=f_M\left(x,\langle M'\rangle\right)$

הרעיון מאחורי M' הוא שמכונה זו מתנהגת כפי ש־M מתנהגת, במקרה הספציפי שבו x הוא קלט כלשהו אבל y אינו סתם קלט אלא הוא הקידוד של M' עצמה. **הוכחה:** בהינתן M, נבנה מכונות A,B בצורה דומה למה שעשינו קודם. נגדיר פונקציה q כך ש־$q(w)=M_w$ כאשר M_w הפעם היא מכונה שעל הקלט x מחזירה (x,w). לאחר שנגדיר את B, תוגדר A להיות $A=q(\langle B\rangle)$. כלומר, זו תהיה מכונה שעל x כותבת על הסרט $(x,\langle B\rangle)$ ועוצרת.

נגדיר פונקציה $r(\langle A\rangle,\langle B\rangle)=\langle AB\rangle$ שמקבלת קידוד של שתי מ"ט ומחזירה מ"ט שמריצה קודם את A ואז את B על הפלט של A.

כעת, B תהיה מכונה שעל הקלט (x,w) מחשבת את $q(w)$, מפרשת את $q(w), w$ בתור קידודים של מ"ט, ואז מחשבת את $y=r(q(w),w)$ ולבסוף מריצה את M על (x,y).

המכונה $M'=r(\langle A\rangle,\langle B\rangle)$ היא המכונה המבוקשת. אופן פעולתה על הקלט x הוא כדלהלן: ראשית רכיב ה־A של M' משנה את תוכן הסרט אל $(x,\langle B\rangle)$. כעת, רכיב ה־B של M' מחשב את $y=r(q(\langle B\rangle),\langle B\rangle)=r(\langle A\rangle,\langle B\rangle)=\langle M'\rangle$. לבסוף, רכיב ה־$B$ מריץ את M על (x,y), כך שהפלט של M' על x יהיה זהה לפלט של M על $(x,\langle M'\rangle)$, כמבוקש. ■

3 בעיות לא כריעות

3.1 בעיות הכרעה של שפות

עד כה עסקנו במכונות טיורינג שמחשבות פונקציות. אם M היא מ"ט, סימנו את הפונקציה אותה היא מחשבת ב־f_M. זה פותח פתח להגדרה הבאה:

הגדרה 1.3 פונקציה $f:\Sigma^*\to\Sigma^*$ היא **ניתנת לחישוב** אם קיימת מ"ט M כך ש־$f=f_M$.

ברצוננו להוכיח כי קיימות פונקציות שאינן ניתנות לחישוב. זו בפני עצמה טענה פשוטה בזכות **שיקולי ספירה**:

משפט 2.3 קיימות פונקציות שאינן ניתנות לחישוב.

הוכחה: עבור א"ב Σ, עוצמת קבוצת הפונקציות מ־Σ^* אל Σ^* היא $\left|\Sigma^{*\Sigma^*}\right|=|\Sigma^*|^{|\Sigma^*|}=\aleph_0^{\aleph_0}=\aleph$. מצד שני, ראינו כי כל מ"ט M ניתנת לקידוד באמצעות מחרוזת **סופית** $\langle M\rangle$ מעל הא"ב $\{0,1\}$. כלומר, קיימת פונקציה חח"ע מקבוצת המ"ט אל $\{0,1\}^*$. מכיוון ש־$|\{0,1\}^*|=\aleph_0$ קיבלנו כי קיימות רק $\aleph_0$ מ"ט. כל מ"ט מחשבת פונקציה אחת בדיוק, ומכאן שקיימת פונקציה f שאין אף מ"ט M המחשבת אותה. ■

ההוכחה הזו אמנם מסיימת את שאלת **הקיום** של פונקציות שאינן ניתנות לחישוב, אבל זה לא פתרון משביע רצון במיוחד. עדיין אין לנו שום דוגמא קונקרטית לפונקציה כזו. אולי כל הפונקציות שאינן ניתנות לחישוב הן כה מסובכות עד שלא ניתן אפילו **לתאר** אותן בצורה משביעת רצון? כפי שנראה זה לא המצב; מכאן ואילך נתעניין בשאלה אילו פונקציות שנראות לנו פשוטות יחסית הן עדיין לא ניתנות לחישוב.

יהיה לנו נוח במיוחד לדבר על תת־קבוצה פשוטה של פונקציות - כאלו שמחזירות רק 0 או 1 על הקלטים שלהן. על פונקציות כאלו אפשר לחשוב כאילו הן אומרות "כן" ו"לא", ולכן בעצם מגדירות **קבוצה** של מילים - הן עונות "כן" על מילים ששייכות לקבוצה, ו"לא" על מילים שאינן שייכות לקבוצה. קבוצות כאלו נקראות **שפות**:

הגדרה 3.3 **שפה** היא תת־קבוצה $L\subseteq\Sigma^*$ כלשהי (שיכולה להיות סופית או אינסופית).

כדי לפשט את העיסוק בשפות נגדיר סוג מיוחד של מ"ט: מכונות לזיהוי שפות.

הגדרה 4.3 מכונת טיורינג לזיהוי שפות היא מ"ט שקבוצת המצבים הסופיים שלה היא $F=\{q_{acc},q_{rej}\}$.
תהא M מ"ט לזיהוי שפות.
נאמר ש־M **מקבלת** את הקלט x אם M עוצרת על x במצב q_{acc}.
נאמר ש־M **דוחה** את הקלט x אם M עוצרת על x במצב q_{rej}.
אם M אינה עוצרת על x לא נאמר שהיא דוחה/מקבלת את x אלא נאמר פשוט שהיא **אינה עוצרת** על x.

מכונות לזיהוי שפות משמשות להגדרה של שפה, אולם כאן נכנסת הבחנה שתהיה קריטית להמשך: יש הבדל בין מכונה ש**עוצרת לכל קלט**, ובין מכונה שעל חלק מהקלטים פשוט אינה עוצרת. הבחירה השרירותית שאנחנו מבצעים בהגדרה שנציג היא זו: להניח שאם מכונה לא עצרה על קלט, הקלט אינו שייך לשפה שהמכונה מגדירה.

הגדרה 5.3 תהא M מ"ט לזיהוי שפות.

השפה שהמכונה M **מקבלת**, המסומנת $L(M)$, היא שפת כל המילים אותן M מקבלת. דהיינו

$$L(M) \triangleq \{x \in \Sigma^* \mid M \text{ accepts } x\}$$

אם בנוסף לכך M עוצרת לכל קלט, נאמר ש־M **מכריעה** את $L(M)$.

נציג מספר דוגמאות לשפות שקיימת מ"ט שמכריעה אותן:

1. השפה $L = \emptyset$ מוכרעת על ידי המכונה שדוחה כל מילה.
2. השפה $L = \Sigma^*$ מוכרעת על ידי המכונה שמקבלת כל מילה.
3. אם L היא שפה סופית, היא ניתנת להכרעה על ידי מכונת טיורינג שחלק מהקידוד שלה כולל את כל המילים של L (למשל, המכונה מתחילה את ריצתה בלכתוב את כל מילות L על הסרט ואז להשוות אותן עם הקלט).
4. השפה $L = \{1^p \mid p \text{ prime}\}$ ניתנת להכרעה על ידי מכונה שעוברת סדרתית על כל המספרים $1 < k < p$ ובודקת האם k מחלק את p. אם כן ־ עוברת ל־q_{rej}, ואם לא ־ בסוף כל הבדיקות עוברת ל־q_{acc}.
5. השפה $L = \{G \mid G \text{ is connected}\}$ של כל הגרפים הלא מכוונים הקשירים ניתנת להכרעה על ידי מכונה שמריצה DFS על גרף הקלט מצומת שרירותי בו ובודקת האם כל צמתי הגרף היו ישיגים ממנו.

שתי הדוגמאות האחרונות ממחישות את הגישה שבה ננקוט מעתה למכונות טיורינג ־ נתאר באופן לא פורמלי את האלגוריתם שהן מריצות, תוך התבססות על אלגוריתמים מוכרים, ובלי להיכנס לדקויות בסגנון האופן שבו אנו מקודדים גרף ־ כל פרטי המידע הללו אינם רלוונטיים לנו בשלב זה.

הגדרה 6.3 נניח כי $\Sigma = \{0, 1\}$.

- מחלקת השפות $L \subseteq \Sigma^*$ אשר קיימת מכונת טיורינג **המכריעה** אותן מסומנת ב־R.
- מחלקת השפות $L \subseteq \Sigma^*$ אשר קיימת מכונת טיורינג **המקבלת** אותן מסומנת ב־RE.

הדרישה $\Sigma = \{0, 1\}$ אינה קריטית; אנו נוקטים בה כדי להימנע מהגדרה של R, RE שתלויה באלפבית Σ.

אנו אומרים על שפה ב־R שהיא **כריעה** או **רקורסיבית** ועל שפה ב־RE שהיא **כריעה למחצה** או **ניתנת למניה רקורסיבית**. המילה "רקורסיבית" כאן אינה במשמעות הרגילה של המונח, אלא היא לקוחה ממאמרו של קורט גדל ונכון יותר לפרש אותה בתור "ניתנת לחישוב". המשמעות של "מניה רקורסיבית" של שפה L היא שקיים אלגוריתם שמייצר סדרתית את כל מילות L (ויכול לרוץ לנצח אם L אינסופית).

ראינו כבר מספר שפות אשר שייכות ל־R. נעמוד כעת על מספר תכונות פשוטות של מחלקת שפות זו.

טענה 7.3 $\mathrm{R} \subseteq \mathrm{RE}$.

הוכחה: טריוויאלי: על פי הגדרה, כל מכונה אשר **מכריעה** שפה L בפרט **מקבלת** את L. ■

טענה 8.3 R סגורה למשלים. כלומר אם $L \in \mathrm{R}$, גם $\overline{L} = \Sigma^* \backslash L$ מקיימת $\overline{L} \in \mathrm{R}$.

הוכחה: הרעיון בהוכחה הוא להשתמש במכונה M עבור L, אבל להחליף את תפקידי המצבים הסופיים שלה. פורמלית, מכיוון ש־$L \in \mathrm{R}$ קיימת מ"ט M כך ש־$L(M) = L$ ו־M מכריעה את L, ובפרט עוצרת לכל קלט.

נגדיר כעת מכונה $\overline{M}$ הזהה ל־M פרט לכך שכל מעבר אל q_{rej} מוחלף במעבר אל q_{acc}, וכל מעבר אל q_{acc} מוחלף במעבר אל q_{rej}.

כעת, $x \in \overline{L}$ אם ורק אם $x \notin L$ אם ורק אם M לא מקבלת את x.

מכיוון ש־M מכריעה את L, העובדה ש־M לא מקבלת את x משמעותה ש־M בהכרח דוחה את x ־ היא אינה יכולה פשוט לא לעצור על x. כלומר, ריצת M על x מסתיימת ב־q_{rej} ולכן ריצת $\overline{M}$ על x תסתיים ב־q_{acc}, כלומר $x \in L(\overline{M})$.

בדומה, אם $x \notin \overline{L}$ נקבל ש־M בריצתה על x מסיימת ב־q_{acc} ולכן $\overline{M}$ תסיים על q_{rej} ומכאן ש־$x \notin L(\overline{M})$. קיבלנו ש־$\overline{L} = L(\overline{M})$. ■

מדוע ההוכחה לא תעבוד עבור RE? מכיוון שהחלפת מצבים של M כללית לא משפיעה על ההתייחסות של M למילים שעליהן היא אינה עוצרת; גם המכונה ה"הפוכה" עדיין לא תקבל מילים אלו, כך ששפתה לא תהיה שפת המשלים של $L(M)$ אם קיימים קלטים שעליהם M אינה עוצרת. זה אינו קושי שניתן לעקוף; בהמשך נראה כי RE אכן אינה סגורה למשלים.

טענה 9.3 R סגורה לאיחוד. כלומר אם $L_1, L_2 \in \mathrm{R}$ אז $L_1 \cup L_2 \in \mathrm{R}$.

הוכחה: יהיו M_1, M_2 מכונות שמכריעות את L_1, L_2 בהתאמה. נבנה מ"ט M שפועלת כך על קלט x: ראשית מריצה את M_1 על x. אם M_1 קיבלה, M מקבלת; אחרת, M מריצה את M_2 על x ועונה כמוה. קל לראות ש־M מקבלת את x אם ורק אם לפחות אחת מהמכונות M_1, M_2 מקבלת את x ולכן

$$L(M) = L(M_1) \cup L(M_2) = L_1 \cup L_2$$

■

באופן דומה ניתן להוכיח גם כי R סגורה לחיתוך (או להשתמש בכללי דה־מורגן והטענות שכבר הוכחנו):

טענה 10.3 R סגורה לחיתוך.

ראינו קודם כי בהגדרת RE קיימת שרירותיות כלשהי ־ עבור מילים שעליהן המכונה M אינה עוצרת, קבענו כי M אינה מקבלת אותן. איזו מחלקה היינו מקבלים אם היינו נוקטים בגישה ההפוכה? התשובה היא המחלקה coRE, שניתנת להגדרה גם בלי לשנות את ההגדרות הקיימות.

הגדרה 11.3 המחלקה coRE מוגדרת בתור

$$\mathrm{coRE} = \left\{ L \mid \overline{L} \in \mathrm{RE} \right\}$$

טענה 12.3 $L \in \mathrm{coRE}$ אם קיימת מ"ט M כך שלכל קלט x:

- אם $x \notin L$ אז M עוצרת על x במצב q_{rej}.
- אם $x \in L$ אז M עוצרת על x במצב q_{acc} או שאינה עוצרת כלל.

הוכחה: מכיוון ש־$L \in \mathrm{coRE}$ הרי ש־$\overline{L} \in \mathrm{RE}$ ולכן קיימת מ"ט $\overline{M}$ כך ש־$L\left(\overline{M}\right) = \overline{L}$. נבנה M מתוך $\overline{M}$ על ידי החלפת q_{acc}, q_{rej}; מכונה זו תקיים את התכונה המבוקשת. ■

שימו לב כי coRE **אינה** המשלימה של RE. בהחלט ייתכן שיהיו בה שפות ששייכות גם ל־RE, וזה תוכן הטענה הבאה שלנו:

טענה 13.3 $\mathrm{RE} \cap \mathrm{coRE} = \mathrm{R}$. כלומר, שפה ששייכת גם ל־RE וגם ל־coRE היא כריעה.

הוכחה: תהא $L \in \mathrm{RE} \cap \mathrm{coRE}$. נבנה מכונה M שמכריעה את L.
תהא M_1 מכונת ה־RE של L ו־M_2 מכונת ה־coRE של L. המכונה M שלנו על קלט x תריץ **במקביל** את M_1, M_2 על x. כלומר, היא תבצע צעד חישוב אחד של M_1 על x, ואז צעד חישוב אחד של M_2 על x וחוזר חלילה.
אם $x \in L$ אז אחרי מספר סופי של צעדי חישוב M_1 תקבל את x ואם $x \notin L$ אז אחרי מספר סופי של צעדי חישוב M_2 תדחה את x. בכל אחד מהמקרים הללו, M תעצור ותענה כמו המכונה שענתה. קיבלנו ש־M עוצרת תמיד, ומקבלת מילה אם ורק אם המילה ב־L, כך ש־M מכריעה את L כמבוקש. ■

נעבור כעת להצגת דוגמאות לשפות השייכות ל־RE. מכיוון ש־$\mathrm{R} \subseteq \mathrm{RE}$ הרי שכל שפה שהיא ב־R היא דוגמא כזו; כאן נתעניין בדוגמאות שעליהן נראה בהמשך שאינן שייכות ל־R.

שפת בעיית העצירה: נתבונן בשפת הזוגות של מכונה וקלט, כך שהמכונה עוצרת על הקלט:

$$\mathrm{HP} = \{ (\langle M \rangle, x) \mid M \text{ halts on } x \}$$

טענה 14.3 $\mathrm{HP} \in \mathrm{RE}$

הוכחה: מכונה M_{HP} שמקבלת את HP תפעל כך: על קלט $(\langle M\rangle, x)$ תריץ את M על x. אם M סיימה את ריצתה, M_{HP} תעצור ותקבל. אם M אינה מסיימת את ריצתה, מן הסתם גם M_{HP} לא תעצור בשום שלב.

אם $(\langle M\rangle, x) \in \text{HP}$ אז M עוצרת מתישהו על x ולכן M_{HP} בריצתה על קלט זה תעצור ותקבל, ולכן $(\langle M\rangle, x) \in L(M_{HP})$.

אם $(\langle M\rangle, x) \notin \text{HP}$ אז M אינה עוצרת על x ולכן M_{HP} בריצתה על קלט זה לא תעצור ולכן הוא לא ייכלל בשפתה, כלומר $(\langle M\rangle, x) \notin L(M_{HP})$.

קיבלנו ש-$L(M_{HP}) = \text{HP}$, כמבוקש. ■

ההוכחה לעיל פשוטה למדי, אבל מכיוון שנשתמש בטכניקה זו של "להריץ מכונה ולעשות משהו אם הוא סיימה, אחרת גם אנחנו רצים לנצח" שוב ושוב הדגמנו אותה כאן בפירוט. כזכור, פשטות ההוכחה נובעת מההשקעה שנדרשה בהוכחת קיום מכונה אוניברסלית שמאפשרת "להריץ" מכונות שנתונות כקלט.

השפה האוניברסלית L_u: נתבונן בשפה

$$L_u = \{(\langle M\rangle, x) \mid M \text{ accepts } x\}$$

שפה זו דומה מאוד ל-HP פרט לכך שאם מריצים את M על x, בסיום ריצת M יש לוודא שהיא נכנסה ל-q_{acc} ולא ל-q_{rej}. מכאן נקבל:

טענה 15.3 $L_u \in \text{RE}$

שפת האלכסון L_D נתבונן בשפה

$$L_D = \{\langle M\rangle \mid M \text{ accepts } \langle M\rangle\} = \{\langle M\rangle \mid \langle M\rangle \in L(M)\}$$

שפה זו היא מעין מקרה פרטי של L_u, כאשר במקום שני קלטים, הקלט השני x הוא במובלע זהה לקלט הראשון. אפשר להוכיח ישירות ש-$L_D \in \text{RE}$ אולם נראה זאת בהמשך בדרך עקיפה כדי להדגים טכניקה כללית יותר.

הסיבה לעניין שלנו בשפה הזו הוא בהפניה העצמית שנמצאת בהגדרתה, שפותחת לנו פתח להוכחה ששפה היא לא כריעה:

נשים לב לכך ש-$\overline{L_D} = \{\langle M\rangle \mid \langle M\rangle \notin L(M)\}$ (כזכור, בגישה שלנו כל מחרוזת מהווה קידוד חוקי של מ"ט - אולי של M_{stam} שלא עוצרת על אף קלט - ולכן ניתן לקחת כך משלים של שפת קידודי מכונות ולקבל שפה של קידודי מכונות).

משפט 16.3 $\overline{L_D} \notin \text{RE}$.

הוכחה: נניח כי קיימת מ"ט M כך ש-$L(M) = \overline{L_D}$. נשאלת השאלה - האם $\langle M\rangle \in L(M)$ או לא?

- נניח כי $\langle M\rangle \in L(M)$. אז על פי הגדרת השפה L_D נובע ש-$\langle M\rangle \in L_D$. מכאן על פי הגדרת משלים ש-$\langle M\rangle \notin \overline{L_D}$, אבל מכיוון ש-$\overline{L_D} = L(M)$ קיבלנו ש-$\langle M\rangle \notin L(M)$ - סתירה להנחה ממנה התחלנו.
- נניח כי $\langle M\rangle \notin L(M)$. אז על פי הגדרת השפה $\overline{L_D}$, $\langle M\rangle \in \overline{L_D}$. אבל מכיוון ש-$L(M) = \overline{L_D}$ נקבל ש-$\langle M\rangle \in L(M)$, בסתירה להנחה ממנה התחלנו.

מכיוון שבכל אחד משני המקרים האפשריים הגענו לסתירה, הרי שעצם ההנחה שקיימת M כך ש-$L(M) = \overline{L_D}$ מובילה לסתירה. מכאן שלא קיימת M כזו, ולכן $\overline{L_D} \notin \text{RE}$. ■

ההוכחה לעיל דומה מאוד באופייה אל **הפרדוקס של ראסל**, שעוסק בקבוצת כל הקבוצות שאינן איבר של עצמן, ומראה שקבוצה כזו תהיה חייבת להיות בו זמנית איבר של עצמה ולא איבר של עצמה, ולכן אינה יכולה להתקיים. כך גם במקרה הנוכחי - מ"ט M שמקבלת את $\overline{L_D}$ תהיה חייבת לענות בו זמנית "כן" ו"לא" על $\langle M\rangle$ ולכן אינה יכולה להתקיים.

מסקנה 17.3 $L_D \notin \text{R}$

הוכחה: אם היה מתקיים $L_D \in \text{R}$ אז מסגירות R למשלים היינו מקבלים $\overline{L_D} \in \text{R} \subseteq \text{RE}$, בסתירה לכך ש-$\overline{L_D} \notin \text{RE}$. ■

3.2 רדוקציות

3.2.1 הגדרה

נזכיר את השפה L_u:

$$L_u = \{(\langle M\rangle, x) \mid M \text{ accepts } x\}$$

אמרנו כי L_D היא מעין "מקרה פרטי" של L_u, ולכן אם $L_D \notin \mathrm{R}$ ניתן לצפות לכך שיתקיים גם $L_u \notin \mathrm{R}$. כיצד נראה זאת פורמלית?

טיעון אפשרי לדוגמא הוא זה: נניח כי קיימת מ"ט M_u אשר מכריעה את L_u. נבנה מכונה M_D שתכריע את L_D באופן הבא: על קלט $\langle M\rangle$, המכונה M_D תריץ את M_u על הקלט $(\langle M\rangle, \langle M\rangle)$ ותענה כמוה. אם M_u קיבלה את הקלט, המשמעות היא ש־M מקבלת את $\langle M\rangle$ ולכן $\langle M\rangle \in L_D$; אם לעומת זאת M_u דחתה את הקלט המשמעות היא ש־M אינה מקבלת את $\langle M\rangle$ ולכן $\langle M\rangle \notin L_D$. קיבלנו ש־$M_D$ אכן עוצרת לכל קלט עם התשובה הנכונה, מה שמוכיח ש־M_D מכריעה את L_D ולכן $L_D \in \mathrm{R}$. מכיוון שאנו יודעים שדבר זה אינו נכון, גם ההנחה ש־$L_u \in \mathrm{R}$ היא שגויה.

נעקוב אחרי המהלך הלוגי שביצענו כאן:

- הנחנו ש־L_u כריעה.
- לקחנו קלט שאנו רוצים לבדוק את שייכותו ל־L_D.
- **המרנו** את הקלט הזה לקלט שאנחנו רוצים לבדוק את שייכותו ל־L_u.
- ענינו על הקלט המקורי את אותה תשובה שהמכונה עבור L_u עונה על הקלט החדש

התהליך הזה, שבו ממירים קלט לבעיה א' בקלט לבעיה ב' כך שהתשובה עבור ב' זהה לתשובה שאמורה להתקבל עבור א', מכונה **רדוקציה** והוא הכלי המרכזי שלנו בהוכחה שבעיות הן בלתי כריעות.

הגדרה 18.3 יהיו $L_1, L_2 \subseteq \Sigma^*$ שפות כלשהן. **רדוקציה** מ־L_1 אל L_2 היא פונקציה **מלאה וניתנת לחישוב** $f : \Sigma^* \to \Sigma^*$ המקיימת

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

אם קיימת רדוקציה מ־L_1 אל L_2 מסמנים זאת $L_1 \le L_2$.

נדגיש מספר נקודות מבלבלות הנוגעות לרדוקציות:

- רדוקציה **אינה** פונקציה $f : L_1 \to L_2$; היא מוגדרת **לכל מילה** ב־Σ^*, כולל אלו שאינן ב־L_1 (כלומר, היא פונקציה **מלאה**). האינטואיציה היא שאנחנו משתמשים ברדוקציות בדיוק כדי לבדוק האם מילה x שייכת ל־L_1 או לא, על ידי המרה של בדיקה זו בבדיקה האם $f(x)$ שייכת ל־L_2.
- רדוקציה אינה חייבת להיות חד־חד־ערכית או על; נראה דוגמאות מפורשות לכך בהמשך.
- הדרישה לכך ש־f תהיה ניתנת לחישוב היא קריטית; בלעדיה, קיימת רדוקציה כמעט בין כל זוג שפות אפשרי.

3.2.2 דוגמאות

דוגמא 1 ראינו כבר בצורה לא פורמלית רדוקציה $L_D \le L_u$; פורמלית, הרדוקציה מוגדרת באמצעות הפונקציה $f(\langle M\rangle) = (\langle M\rangle, \langle M\rangle)$ שהיא בוודאי ניתנת לחישוב כי בסך הכל מתבצע בה שכפול של הקלט.

דוגמא 2 נזכיר את השפה HP:

$$\mathrm{HP}=\{(\langle M\rangle,x)\ \mid M\text{ halts on }x\}$$

נראה רדוקציה $\mathrm{HP}\le L_u$. הרדוקציה תוגדר כך: $f(\langle M\rangle,x)=(\langle M'\rangle,x)$ כך ש־M' זהה ל־M למעט שינוי כל מעבר שמוביל אל q_{rej} למעבר שמוביל אל q_{acc}. שינוי כזה הוא **ניתן לחישוב** - פשוט עוברים על הקידוד של $\langle M\rangle$ ומשנים את המקומות המתאימים (שינוי כזה יכונה "פעולת קומפילציה פשוטה" על ידינו בהמשך).
נראה את נכונות הרדוקציה:

- אם $(\langle M\rangle,x)\in\mathrm{HP}$ אז M עוצרת על x (במצב q_{acc} או במצב q_{rej}). מכאן ש־M' עוצרת גם היא על x, אבל היא יכולה לעצור רק במצב q_{acc} ולכן M' מקבלת את x, כך ש־$(\langle M'\rangle,x)\in L_u$.
- אם $(\langle M\rangle,x)\notin\mathrm{HP}$ אז M אינה עוצרת על x ולכן גם M' אינה עוצרת על x ובפרט אינה מקבלת אותו, כך ש־$(\langle M'\rangle,x)\notin L_u$.

נראה כעת רדוקציה **בכיוון השני**, $L_u\le\mathrm{HP}$. הרדוקציה תוגדר כך: $f(\langle M\rangle,x)=(\langle M'\rangle,x)$ כך ש־M' זהה ל־M למעט העובדה שבמקום מעבר אל q_{rej}, המכונה עוברת אל מצב של לולאה אינסופית (מצב שבו המכונה לא משנה כלום ונשארת באותו מצב). כלומר, M' עוצרת על קלט אם ורק אם M מקבלת אותו, מה שמראה את נכונות הרדוקציה.

דוגמא 3 לכל שפה $L\in\mathrm{R}$ קיימת רדוקציה אל HP: תהא M_1 מכונה שעוצרת על כל קלט ו־M_2 מכונה שאינה עוצרת על אף קלט, אז הרדוקציה $L\le\mathrm{HP}$ שלנו תפעל באופן הבא: עבור קלט x, ראשית המכונה שמחשבת את פונקציית הרדוקציה תפעיל את המכונה שמכריעה את L על x ותבדוק מה תשובתה. אם $x\in L$ אז המכונה תחזיר $(\langle M_1\rangle,\varepsilon)$ ואילו גם $x\notin L$ אז היא תחזיר $(\langle M_2\rangle,\varepsilon)$.

קל לבדוק את נכונות הרדוקציה; החלק הלא טריוויאלי בבניה הוא אופן חישוב פונקציית הרדוקציה עצמה, שלא כולל שינוי קל בקלט אלא ביצוע חישוב מורכב עליו (בדיקת שייכות ל־L) והחזרת אחת משתי תשובות מוכנות מראש בהתאם לתוצאה.

נשים לב לכך שאין ל־HP חשיבות גדולה בהקשר זה. כל שפה L' כך ש־$L'\ne\emptyset,\Sigma^*$ תעבוד, כי עבור כל שפה כזו קיימות מילים a,b כך ש־$a\in L'$ ו־$b\notin L'$ והרדוקציה שתיארנו לעיל תעבוד, עם החזרת a במקרה הראשון ו־b במקרה השני.

דוגמא 4 כל שפה L ניתנת לרדוקציה לעצמה, $L\le L$, על ידי הפונקציה $f(x)=x$.

אם $L_1\le L_2$ וגם $L_2\le L_3$ אז נובע מכך ש־$L_1\le L_3$. נוכיח זאת: תהא f הרדוקציה $L_1\le L_2$ ו־g הרדוקציה $L_2\le L_3$, אז ההרכבה gf היא רדוקציה $L_1\le L_3$:

ראשית, gf ניתנת לחישוב על ידי מכונה שראשית מפעילה את המכונה של f על הקלט, ואז מפעילה את המכונה של g על הפלט של f.

שנית, $x\in L_1$ אם ורק אם $f(x)\in L_2$ אם ורק אם $g(f(x))\in L_3$, כנדרש.

התכונה $L\le L$ נקראת **רפלקסיביות** והתכונה $L_1\le L_3 \Leftarrow L_1\le L_2\wedge L_2\le L_3$ נקראת **טרנזיטיביות**. שתי תכונות אלו מאפיינות גם את יחס הסדר הרגיל של מספרים $\le$, ומכאן השימוש בסימן $\le$ שנפוץ במתמטיקה לתיאור יחסי סדר באופן כללי. עם זאת, קיים הבדל מהותי אחד: במספרים רגילים, $a\le b$ וגם $b\le a$ גורר $a=b$; תכונה זו נקראת **אנטי־סימטריה**. תכונה זו **אינה** מתקיימת עבור רדוקציות. למשל, ראינו כבר כי $L_u\le\mathrm{HP}$ וגם $\mathrm{HP}\le L_u$ אבל אלו שפות שונות (במתמטיקה, יחס רפלקסיבי וטרנזיטיבי נקרא **קדם־סדר**).

דוגמא 5 אם $L_1\le L_2$ אז גם $\overline{L_1}\le\overline{L_2}$.
אותה רדוקציה בדיוק שמראה את $L_1\le L_2$ תראה גם את $\overline{L_1}\le\overline{L_2}$. זאת משום ש:

$$x\in\overline{L_1}\iff x\notin L_1\iff f(x)\notin L_2\iff f(x)\in\overline{L_2}$$

3.2.3 משפט הרדוקציה

הצגנו רדוקציות בתור אמצעי להכריע שפה אחת במקרה שבו אנחנו כבר יודעים להכריע שפה אחרת. ננסח זאת פורמלית:

משפט 19.3 תהיינה L_1, L_2 שפות כך ש־$L_1 \leq L_2$

- אם $L_2 \in \mathrm{R}$ אז $L_1 \in \mathrm{R}$.
- אם $L_2 \in \mathrm{RE}$ אז $L_1 \in \mathrm{RE}$.

הוכחה: תהא M המכונה שמחשבת את הרדוקציה $L_1 \leq L_2$ ותהא M_2 מכונה שמכריעה/מקבלת את L_2. נבנה מכונה M_1 שמכריעה/מקבלת את L_1: M_1 על קלט x תפעיל את M על x, תקבל פלט y תפעיל את M_2 על y ואם M_2 עצרה, תענה כמוה.

$x \in L_1$ אם ורק אם $M(x) \in L_2$ (על פי הגדרת רדוקציה), אם ורק אם M_2 מקבלת את $M(x)$ (על פי הגדרת שפה של מ"ט), ולכן $L(M_1) = L_1$. בנוסף לכך, אם M_2 עוצרת לכל קלט, אז גם M עוצרת לכל קלט (כי החישוב של M מסתיים לכל קלט, והחישוב של M_2 מסתיים לכל קלט) ולכן במקרה זה M_1 מכריעה את L_1. ■

אנו בדרך כלל משתמשים במשפט הרדוקציה דווקא כדי להראות ששפה **איננה** ב־R או ב־RE על ידי לקיחת ניסוח שקול של משפט הרדוקציה:

משפט 20.3 תהיינה L_1, L_2 שפות כך ש־$L_1 \leq L_2$

- אם $L_1 \notin \mathrm{R}$ אז $L_2 \notin \mathrm{R}$.
- אם $L_1 \notin \mathrm{RE}$ אז $L_2 \notin \mathrm{RE}$.

השימושיות של רדוקציות היא גדולה מאוד, אבל קל להתבלבל ולבצע רדוקציה "בכיוון הלא נכון" כשרוצים להוכיח ששפה איננה כריעה. כלל האצבע שיש לזכור הוא: אם רוצים להראות ששפה אינה כריעה, צריך לבצע רדוקציה **אליה** משפה שכבר ידוע שאינה כריעה - להראות שהשפה שלנו קשה **יותר** מאשר השפה שכבר מוכרת.

למרות שאנו על פי רוב מתעניינים פחות ב־coRE, משפט הרדוקציה מאפשר לנו להוכיח אי שייכות אליה באותה המידה:

טענה 21.3 תהיינה L_1, L_2 שפות כך ש־$L_1 \leq L_2$. אם $L_1 \notin \mathrm{coRE}$ אז $L_2 \notin \mathrm{coRE}$.

הוכחה: אם $L_1 \notin \mathrm{coRE}$ אז $\overline{L_1} \notin \mathrm{RE}$. הרדוקציה $L_1 \leq L_2$ מראה ש־$\overline{L_1} \leq \overline{L_2}$, כך שניתן להסיק ש־$\overline{L_2} \notin \mathrm{RE}$ ומכאן ש־$L_2 \notin \mathrm{coRE}$. ■

נעבור למספר דוגמאות.

דוגמא 1 ראינו כבר כי $\overline{L_D} \notin \mathrm{RE}$ וכי $L_D \leq L_u$. מכאן נסיק כי $\overline{L_D} \leq \overline{L_u}$ ולכן $\overline{L_u} \notin \mathrm{RE}$, בנוסף לכך ש־$L_u \notin \mathrm{R}$.
בדומה, $L_u \leq \mathrm{HP}$ מאפשר לנו להסיק כי $\mathrm{HP} \notin \mathrm{R}$.
מכיוון ש־$\mathrm{HP} \notin \mathrm{R}$ אבל $\mathrm{HP} \in \mathrm{RE}$ נוכל להסיק כי $\overline{\mathrm{HP}} \notin \mathrm{RE}$: שכן אם היה מתקיים $\overline{\mathrm{HP}} \in \mathrm{RE}$ היינו מקבלים $\mathrm{HP} \in \mathrm{coRE}$ ולכן $\mathrm{HP} \in \mathrm{RE} \cap \mathrm{coRE} = \mathrm{R}$, בסתירה לכך ש־$\mathrm{HP} \notin \mathrm{R}$.
נשים לב כי $\overline{\mathrm{HP}}$ הוגדרה בתור המשלימה של HP, ולכן היא כוללת שני סוגי איברים: זוגות $(\langle M \rangle, x)$ כך ש־M אינה עוצרת על x; ומחרוזות w שאינן קידוד חוקי כלל של מכונה וקלט. כדי לפשט את הסימונים שלנו נניח כי מחרוזות כאלו מקודדות את הזוג $(\langle M_{stam} \rangle, \varepsilon)$ של המכונה שאינה עוצרת על אף קלט ושל המילה הריקה.

דוגמא 2 נתבונן בשפה $L_\varepsilon = \{\langle M \rangle \mid \varepsilon \in L(M)\}$ של כל המכונות אשר מקבלות את המילה הריקה.
בבירור שפה זו שייכת ל־RE; בהינתן $\langle M \rangle$ ניתן פשוט להריץ אותה על ε ולקבל אם ורק אם M קיבלה.
עם זאת, השפה אינה שייכת ל־R ונראה זאת באמצעות רדוקציה $\mathrm{HP} \leq L_\varepsilon$.
הרדוקציה תוגדר כך: $f(\langle M \rangle, x) = \langle M_x \rangle$ כך ש־M_x היא מכונה שעל קלט w:

- מריצה את M על x.
- אם M עצרה, מקבלת.

כלומר, האופן שבו M_x פועלת אינו תלוי כלל בקלט שלה; היא כוללת בתוכה את M, x ומתעניינת רק בתוצאה של הרצת M על x. בשל כך, יש שתי דרכים אפשריות שבהן M_x עשויה לפעול:

- אם M עוצרת על x, אז M_x תקבל כל קלט.
- אם M אינה עוצרת על x אז M_x לא תעצור על אף קלט.

בפרט, M עוצרת על x אם ורק אם $\varepsilon \in L(M_x)$, מה שמוכיח את תקפות הרדוקציה ולכן, מכיוון ש־$\mathrm{HP} \notin \mathrm{R}$, מראה ש־$L_\varepsilon \notin \mathrm{R}$.

דוגמא 3 נתבונן בשפה $L_{EQ} = \{(\langle M_1\rangle, \langle M_2\rangle) \mid L(M_1) = L(M_2)\}$ של זוגות של מכונות בעלות אותה שפה. קל להציג רדוקציה מ־HP לשפה זו, תוך שימוש באבחנה שראינו קודם ־ שאם M עוצרת על x, אז M_x מקבלת כל קלט.

נסמן ב־M_{Σ^*} מ"ט שעוברת מייד ל־q_{acc} על כל קלט, והרדוקציה $\mathrm{HP} \le L_{EQ}$ תוגדר על ידי

$$(\langle M\rangle, x) \mapsto (\langle M_x\rangle, \langle M_{\Sigma^*}\rangle)$$

תקפות הרדוקציה נובעת מכך שאם M עוצרת על x אז $L(M_x) = \Sigma^*$ ואילו אם M אינה עוצרת על x אז $L(M_x) = \emptyset$.

מכאן נקבל באמצעות משפט הרדוקציה ש־$L_{EQ} \notin \mathrm{R}$.

נוכל להראות גם שמתקיים $L_{EQ} \notin \mathrm{RE}$ באמצעות רדוקציה מ־$\overline{\mathrm{HP}}$. הרדוקציה תהיה דומה מאוד ותשתמש באבחנה שכבר ראינו:

$$(\langle M\rangle, x) \mapsto (\langle M_x\rangle, \langle M_\emptyset\rangle)$$

כאשר $M_\emptyset$ היא מכונה שדוחה מייד כל קלט ולכן $L(M_\emptyset) = \emptyset$.

האם הרדוקציה הראשונה שלנו הייתה מיותרת? לא, שכן היא מראה גם כי $L_{EQ} \notin \mathrm{coRE}$, כך ש־$L_{EQ}$ שלנו אינה שייכת ל־$\mathrm{RE} \cup \mathrm{coRE}$ ־ השפה הראשונה שראינו שמקיימת זאת.

3.3 משפט רייס

משפט רייס עוסק בסיטואציה הבאה: נניח ששפה כלשהי נתונה לנו באמצעות מכונת טיורינג; מה אנחנו יכולים להגיד על השפה? התשובה היא "כלום". ליתר דיוק ־ אין לנו אלגוריתם שמקבל מכונת טיורינג ומכריע את השאלה האם השפה של אותה מכונה מקיימת תכונה לא טריוויאלית כלשהי.

באופן כללי, אם אנחנו מסכימים על שיטה כלשהי לייצוג שפות, ייתכן שנוכל לחלץ מידע על השפה מתוך הייצוג שלה. למשל, אם בשיטת הייצוג שלנו כל שפה סופית מיוצגת על ידי רשימת כל המילים שבה, בעוד שבשפות אינסופיות משתמשים בקיצור או בסימן $\ldots$, אז ממבט בייצוג של השפה נוכל להבין אם היא סופית או אינסופית. עבור מכונות טיורינג אפילו זה יהיה בלתי אפשרי.

נחדד את הכוונה שלנו באמצעות הגדרה פורמלית:

הגדרה 22.3 **תכונה** של שפות ב־RE היא תת־קבוצה $S \subseteq \mathrm{RE}$. נאמר שתכונה S היא **טריוויאלית** אם $S = \emptyset$ או $S = \mathrm{RE}$.

כשנגדיר תכונה בפועל לא נטרח לציין את העובדה שהשפות בתכונה הן ב־RE כדי למנוע סירבול. למשל, "להיות שפה אינסופית" היא תכונה לא טריוויאלית של שפות ב־RE; "להכיל את ε" היא תכונה לא טריוויאלית של שפות ב־RE, וכדומה.

משפט רייס מבהיר לנו שהתקווה לבדוק אם שפה של מ"ט נתונה מקיימת תכונה S היא משוללת יסוד:

משפט 23.3 (משפט רייס): תהא S תכונה לא טריוויאלית של שפות ב־RE. נסמן

$$L_S = \{\langle M\rangle \mid L(M) \in S\}$$

אז $L_S \notin \mathrm{R}$.

אם בנוסף לכך $\emptyset \in S$ אז $L_S \notin \mathrm{RE}$.

מדוע חשוב שהתכונה S תהיה לא טריוויאלית? שכן אם $S=\emptyset$, כלומר **כל** שפה ב־RE אינה מקיימת את התכונה, אז L_S מוכרעת על ידי מכונה שתמיד אומרת "לא". בדומה, אם $S=\mathrm{RE}$ אז L_S מוכרעת על ידי מכונה שתמיד אומרת "כן".

הוכחת משפט רייס היא בבסיסה רדוקציה הדומה לאלו שראינו עד כה:

הוכחה: נניח ש־$\emptyset\notin S$. במקרה זה נציג רדוקציה $\mathrm{HP}\le L_S$, מה שיוכיח ש־$L_S\notin\mathrm{R}$.
מכיוון ש־S היא תכונה לא טריוויאלית, קיימת שפה $L\in S$. תהא M_L מ"ט כך ש־$L\left(M_L\right)=L$.
כעת הרדוקציה $\mathrm{HP}\le L_S$ שלנו תוגדר כך: $\left(\langle M\rangle,x\right)\mapsto\langle M_x\rangle$ כאשר M_x היא מכונה שעל קלט w פועלת כך:

- מריצה את M על x.
- אם M עצרה על x, מריצה את M_L על w ועונה כמוה.

נבדיל כעת בין שני מקרים:

- אם $\left(\langle M\rangle,x\right)\in\mathrm{HP}$ אז M עוצרת על x, ולכן M_x בריצתה על w תמיד מריצה את M_L על w ועונה כמוה, ולכן $L\left(M_x\right)=L\left(M_L\right)\in S$.
- אם $\left(\langle M\rangle,x\right)\notin\mathrm{HP}$ אז M אינה עוצרת על x, ולכן M_x בריצתה על w תמיד אינה עוצרת, ולכן $L\left(M_x\right)=\emptyset\notin S$.

ראינו את תקפות הרדוקציה $\mathrm{HP}\le L_S$ במקרה שבו $\emptyset\notin S$.
נעבור כעת אל המקרה $\emptyset\in S$. מקרה זה יהיה דומה לקודמו, אך הפעם נראה רדוקציה $\overline{\mathrm{HP}}\le L_S$; רדוקציה זו תראה לנו ש־$L_S\notin\mathrm{RE}$, כמבוקש.
ראשית, מכיוון ש־S אינה טריוויאלית, קיימת L כך ש־$L\notin S$ (שימו לב להיפוך התפקידים ביחס לחלק הקודם של ההוכחה). כמקודם, תהא M_L מכונה כך ש־$L\left(M_L\right)=L$.
הרדוקציה שלנו $\left(\langle M\rangle,x\right)\mapsto\langle M_x\rangle$ תוגדר בדיוק כמו קודם. נשים לב לתקפות שלה:

- אם $\left(\langle M\rangle,x\right)\in\overline{\mathrm{HP}}$ אז M אינה עוצרת על x, ולכן M_x בריצתה על w תמיד אינה עוצרת, ולכן $L\left(M_x\right)=\emptyset\in S$.
- אם $\left(\langle M\rangle,x\right)\notin\overline{\mathrm{HP}}$ אז M עוצרת על x, ולכן M_x בריצתה על w תמיד מריצה את M_L על w ועונה כמוה, ולכן $L\left(M_x\right)=L\left(M_L\right)\notin S$.

זה מסיים את הוכחת המקרה השני. ■

משפט רייס הוא כלי יעיל מאוד להוכחה ששפות רבות אינן ב־R או ב־RE. נראה מספר דוגמאות לכך.

דוגמא 1 התכונה $S=\{L\in\mathrm{RE}\mid\varepsilon\in L\}$ היא תכונה של שפות ב־RE שאינה טריוויאלית (כי $\emptyset\notin S$ אבל $\Sigma^*\in S$) ומכאן ש־$L_S=L_\varepsilon$ אינה ב־R. זה לא מחדש לנו הרבה כי כבר ראינו רדוקציה מפורשת כזו שלמעשה הייתה זהה לזו שניתנת בהוכחה הכללית של משפט רייס (באותה רדוקציה מפורשת, השפה L שבה השתמשנו הייתה Σ^*). עם זאת, אנו רואים כאן כי גם עבור מילים שונות מ־ε, או כל תת־קבוצה אפשרית של מילים שאנו דורשים שכולן ישתייכו ל־L, עדיין נקבל שהשפה אינה ב־R.

דוגמא 2 נתבונן בתכונה $S=\{\Sigma^*\}$, שמניבה את השפה $L_{\Sigma^*}=\{\langle M\rangle\mid L\left(M\right)=\Sigma^*\}$. משפט רייס מראה לנו מייד כי $L_{\Sigma^*}\notin\mathrm{R}$. עם זאת, זו תוצאה חלשה יחסית למה שניתן להוכיח על השפה בדרכים אחרות: בפועל, $L_{\Sigma^*}\notin\mathrm{RE}\cup\mathrm{coRE}$. אך משפט רייס אינו מאפשר לנו להוכיח זאת, שכן $\emptyset\notin S$ ולכן לא ניתן להסיק $L_{\Sigma^*}\notin\mathrm{RE}$ (אם כי במקרה זה המשפט אכן מראה את הטענה שפחות מעניינת אותנו, $L_{\Sigma^*}\notin\mathrm{coRE}$). בהמשך נראה טכניקות שמאפשרות לנו להוכיח את הטענה המורכבת יותר.

דוגמא 3 נגדיר שלוש שפות:

$$L_{\le3}=\{\langle M\rangle\mid\left|L\left(M\right)\right|\le3\}$$
$$L_{=3}=\{\langle M\rangle\mid\left|L\left(M\right)\right|=3\}$$
$$L_{\ge3}=\{\langle M\rangle\mid\left|L\left(M\right)\right|\ge3\}$$

התכונות המתאימות לשפות אלו הן:

$$S_{\le 3}=\{L\in \mathrm{RE} \mid |L|\le 3\}$$
$$S_{=3}=\{L\in \mathrm{RE} \mid |L|=3\}$$
$$S_{\ge 3}=\{L\in \mathrm{RE} \mid |L|\ge 3\}$$

תכונות אלו הן בבירור לא טריוויאליות ולכן כל שלוש השפות אינן ב־R. בנוסף לכך ממשפט רייס ניתן להסיק כי $L_{\le 3}\notin \mathrm{RE}$, כי $\emptyset\in S_{\le 3}$.

עבור השפות האחרות, $L_{\ge 3}\in \mathrm{RE}$ אך נזדקק לטכניקות נוספות כדי להראות זאת, ואילו $L_{=3}\notin \mathrm{RE}$ ונוכל להראות זאת כעת.

כדי להראות כי $L_{=3}$ נשתמש ברדוקציה $\overline{\mathrm{HP}}\le L_{=3}$ שדומה לרדוקציה בה משתמשים במשפט רייס אך מחוכמת מעט יותר; זה מראה את האופן השרירותי במידת מה שבו הגדרנו את משפט רייס, כי את השפה $\emptyset$ היינו יכולים להחליף בשפות רבות נוספות. קיימת למשפט רייס גרסה מלאה יותר, של "אם ורק אם", שבה מנוסח קריטריון מורכב שמצליח להתייחס לכל השפות האפשריות הללו.

הרדוקציה תוגדר כך: $(\langle M\rangle ,x)\mapsto\langle M_x\rangle$ כך ש־M_x על קלט w פועלת כך:

- אם $w\in\{\varepsilon,0,1\}$, אז M_x מקבלת מייד.
- אחרת, M_x מריצה את M על x ואם M עצרה ־ מקבלת.

כתוצאה מכך, יש שתי אפשרויות:

$$L(M_x)=\begin{cases}\{\varepsilon,0,1\} & (\langle M\rangle ,x)\in\overline{\mathrm{HP}}\\ \Sigma^{*} & (\langle M\rangle ,x)\notin\overline{\mathrm{HP}}\end{cases}$$

בבירור, רדוקציה זו מראה ש־$L_{=3}\notin \mathrm{RE}$, וכי היינו יכולים להחליף את $\emptyset$ במשפט רייס גם בשפה $\{\varepsilon,0,1\}$.

3.4 הרצה מבוקרת

עד כה הרדוקציות שלנו היו כולן מאותו סגנון: בהינתן מכונה M וקלט x, בנינו מכונה M_x שדבר ראשון הריצה את M על x ואולי עשתה אז דברים נוספים. כלומר, הפעולה הראשונה של המכונה שלנו הייתה להפוך באופן זמני למכונה אחרת, כך שהמשך הריצה שלה היה תלוי בכך שהמכונה האחרת תסיים. זו גישה נאיבית למדי, שאינה מנצלת את מלוא היכולות של מכונת טיורינג.

כזכור, כאשר בנינו את המכונה האוניברסלית, ראינו כי יש לנו שליטה מלאה על אופן הרצת המכונה ־ אנחנו מייצרים קונפיגורציות בצורה סדרתית, ויכולים בכל עת לקחת הפסקה מייצור הקונפיגורציות, לשנות את הקונפיגורציות כאוות נפשנו, וכדומה. בפרט, אנחנו מסוגלים לבצע **מספר חישובים במקביל** ואנחנו גם יכולים לקבוע שנבצע חישוב מסויים רק למשך מספר **מוגבל** של צעדים. את היכולות הנוספות הללו אנחנו מכניסים תחת השם **הרצה מבוקרת** שכן במקום להריץ בצורה "חופשית" את M על x אנחנו מכניסים ממד של **בקרה** על האופן שבו הריצה הזו מתבצעת (הדבר דומה למנגנון ה־interrupt־ים שקיים במחשבים מודרניים, או פשוט להרצה באמצעות דיבאגר).

דוגמא ־ הרצה על אינסוף קלטים במקביל נוכיח כי $L_{\ge 3}\in \mathrm{RE}$ על ידי מכונה שמקבלת את השפה. הרעיון של המכונה יהיה להריץ את מכונת הקלט M "במקביל" על כל אינסוף המילים האפשריות; אם בשלב כלשהו של ההרצה המקבילית הזו תתקבלנה שלוש מילים, אפשר לעצור ולקבל את $\langle M\rangle$. אחרת, בהכרח שפת M כוללת פחות מ־3 מילים ולכן ריצה לנצח שמשמעותה אי־קבלת $\langle M\rangle$ היא אכן מה שצריך להתרחש פה.

כיצד ניתן לרוץ על יותר מקלט אחד בו זמנית? במקום לשמור על הסרט קונפיגורציה אחת בכל פעם, אפשר לשמור עליו סדרה (סופית) של קונפיגורציות, כל אחת שמתאימה לריצה על קלט אחר, ובכל פעם לקדם את אחת מהקונפיגורציות צעד אחד, כרצוננו. בפועל המכונה יכולה לפעול כך:

- בצעי צעד אחד על הקלט ε
- בצעי שני צעדים על הקלט ε ושני צעדים על הקלט 1
- בצעי שלושה צעדים על הקלט ε, שלושה צעדים על הקלט 1 ושלושה צעדים על הקלט 2

- וכן הלאה

נוכל לסמן $\Sigma^* = \{w_1, w_2, w_3, \ldots\}$ על פי סדר כלשהו כדוגמת הסדר הלקסיקוגרפי ($\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$) ואז האלגוריתם ניתן לתיאור כללי כך: לכל $n = 1, 2, 3, \ldots$, מבוצעים n צעדי חישוב על הקלטים $w_1, w_2, \ldots, w_n$.

נניח כי M מקבלת את הקלט w_t תוך k צעדים. אז בשלב שבו $n = \max\{k, t\}$ מובטח לנו כי M תבצע לפחות k צעדים על הקלט w_t, ולכן נגלה בשלב זה כי M מקבלת את w_t. כך יהיה עבור כל קלט אותו M מקבלת, כך שאם קיימים שלושה קלטים שהיא מקבלת, אנחנו נזהה זאת ונוכל לקבל.

דוגמא - הרצה למספר מוגבל של צעדים נרצה להוכיח כי $L_{\Sigma^*} \notin \mathrm{RE}$. נעשה זאת באמצעות רדוקציה $\overline{\mathrm{HP}} \leq L_{\Sigma^*}$. תחילה נציג שתי רדוקציות כושלות ונבין מה בדיוק השתבש בהן.

הרדוקציה הראשונה היא זו שבה השתמשנו כבר פעמים רבות: $(\langle M \rangle, x) \mapsto \langle M_x \rangle$ כאשר M_x מריצה את M על x ומקבלת אם M עצרה. במקרה זה מקבלים

$$L(M_x) = \begin{cases} \emptyset & (\langle M \rangle, x) \in \overline{\mathrm{HP}} \\ \Sigma^* & (\langle M \rangle, x) \notin \overline{\mathrm{HP}} \end{cases}$$

זה כמעט הפוך ממה שאנחנו רוצים. אנחנו רוצים שדווקא אם $(\langle M \rangle, x) \in \overline{\mathrm{HP}}$ אז $L(M_x) = \Sigma^*$, ואילו אם $(\langle M \rangle, x) \notin \overline{\mathrm{HP}}$ (כלומר, M עוצרת על x) אז $L(M_x) = \emptyset$.

ננסה לתקן את הרדוקציה: כעת M_x תריץ את M על x כמקודם, אבל אם M עצרה אז M_x **תדחה** את הקלט. כלומר, אם $(\langle M \rangle, x) \notin \overline{\mathrm{HP}}$ אז $L(M_x) = \emptyset$. לרוע המזל, אם M אינה עוצרת על x, אז בפרט M_x לעולם לא תגיע לשלב שבו היא מקבלת קלטים, ולכן נקבל $L(M_x) = \emptyset$ גם אם $(\langle M \rangle, x) \in \overline{\mathrm{HP}}$, וזה בוודאי לא מה שרצינו.

לכן ננקוט בתעלול. M_x על הקלט w תפעל כך:

1. תריץ את M על x למשך $|w|$ צעדים.
2. אם במהלך הריצה הזו M עצרה על x, אז M_x **תדחה** את w.
3. אחרת, M_x **תקבל** את w.

אם $(\langle M \rangle, x) \in \overline{\mathrm{HP}}$, אז מובטח לנו ששלב 2 לא יתקיים לעולם - לא משנה למשך כמה צעדים נריץ את M על x, היא לעולם לא תעצור. לכן תמיד נגיע לשלב 3 ותמיד נקבל, כך ש-$L(M_x) = \Sigma^*$ - כפי שרצינו.

אם לעומת זאת $(\langle M \rangle, x) \notin \overline{\mathrm{HP}}$ אז M עוצרת על x אחרי k צעדים בדיוק. אם כן, לכל קלט w כך ש-$|w| < k$ בשלב 2 לא נגיע לכך ש-M עצרה על x ולכן **נקבל** כל קלט כזה; אבל אם $|w| \geq k$ אז בשלב 2 **תמיד** נגיע לכך ש-M עוצרת על x ולכן M_x תדחה. מכאן שבמקרה זה, $L(M_x) = \{w \in \Sigma^* \mid |w| < k\}$, ומכיוון ששפה זו שונה מ-Σ^*, זה מסיים את הוכחת נכונות הרדוקציה.

כיצד ניתן להשתמש בטכניקה שראינו על מנת להתמודד עם שפות נוספות? אותה רדוקציה בדיוק תעבוד גם עבור השפה $L_{inf} = \{\langle M \rangle \mid |L(M)| = \infty\}$, כי אם M עוצרת על x אז השפה של M_x היא סופית ואם אינה עוצרת, אז שפת M_x היא אינסופית.

בשלב 3 של פעולת M_x המכונה אינה **חייבת** לקבל; היא יכולה להריץ מכונה כלשהי על הקלט w ולענות כמוה, כך שנקבל את ההפרדה הבאה: אם $(\langle M \rangle, x) \in \overline{\mathrm{HP}}$ אז נקבל ש-$L(M_x)$ היא שפה כלשהי ב-RE לבחירתנו, ואילו אם $(\langle M \rangle, x) \notin \overline{\mathrm{HP}}$ אז $L(M_x)$ היא שפה כלשהי שאנחנו יודעים שהיא **סופית**. כך למשל אפשר להראות ששפת כל המכונות M שמקבלות בדיוק את כל המילים מאורך זוגי אינה ב-RE; בשלב 3, במקום לקבל את w ראשית נבדוק אם w מאורך זוגי ורק אם כן, נקבל.

3.5 חישוב פונקציות

התחלנו עם המושג של פונקציה ניתנת לחישוב ואז עברנו לעסוק בשפות. כעת נראה את הקשר בין שני המושגים. נזכיר את ההגדרות הבסיסיות שלנו:

הגדרה 24.3 פונקציה $f : \Sigma^* \to \Gamma^*$ נקראת **מלאה** אם היא מוגדרת לכל קלט. היא נקראת **ניתנת לחישוב** אם קיימת מ"ט M כך ש-$f_M = f$.

בהינתן פונקציה f, נגדיר את השפה המתארת אותה: $L_f = \{(x, y) \mid f(x) = y\}$. שימו לב כי $(x, y) \in L_f$ פירושו ש-f מוגדרת על x **וגם** $f(x) = y$; אם f אינה מוגדרת על x, אז לא יופיע זוג שבו x הוא האיבר השמאלי.

משפט 25.3 תהא f פונקציה

- f ניתנת לחישוב אם ורק אם $L_f \in \mathrm{RE}$
- אם f מלאה, אז f ניתנת לחישוב אם ורק אם $L_f \in \mathrm{R}$

הוכחה: נניח כי f ניתנת לחישוב באמצעות מכונה M_f. מכונה M עבור L_f תפעל כך על קלט (x,y): ראשית תריץ את M_f על x. אם M_f עצרה, M תשווה את הפלט של M_f ל־y ותקבל רק אם הם שווים. בבירור M אכן מקבלת את L_f, ואם f מלאה אז M_f עוצרת לכל קלט ולכן M תעצור לכל קלט, אז במקרה זה היא **מכריעה** את L_f.

בכיוון השני, אם $L_f \in \mathrm{RE}$ עם מכונה M_f כך ש־$L(M_f) = L_f$ אז מכונה לחישוב f תפעל כך: בהינתן קלט x, המכונה תבצע הרצה מבוקרת של M_f על כל הקלטים מהצורה (x,y) לכל $y \in \Sigma^*$. אם M_f עצרה וקיבלה זוג (x,y), המכונה לחישוב f תעצור ותוציא את y כפלט. ■

בסיוע המשפט ניתן להמיר את השאלה האם פונקציה היא ניתנת לחישוב בשאלה האם שפה שייכת ל־RE, שיכולה להיות קלה יותר למענה בזכות כלי הרדוקציות שברשותנו.

דוגמא: פונקציית גודל השפה נגדיר פונקציה אשר בהינתן מ"ט M מחזירה את גודל השפה של M אם היא סופית, ואינה מוגדרת במקרה שבו השפה אינסופית:

$$f(\langle M\rangle) = \begin{cases} |L(M)| & |L(M)| < \infty \\ \perp & |L(M)| = \infty \end{cases}$$

ניתן להראות באופן ישיר כי f אינה ניתנת לחישוב. נניח כי היא כן ניתנת לחישוב עם מכונה M_f וניעזר בה כדי להוכיח שהשפה $L_\emptyset = \{\langle M\rangle \mid L(M) = \emptyset\}$ שייכת ל־RE למרות שרואים מיידית ממשפט רייס כי $L_\emptyset \notin \mathrm{RE}$.

נבנה מכונה $M_\emptyset$ שפועלת כך על קלט $\langle M\rangle$:

- מריצה את M_f על $\langle M\rangle$.
- אם M_f סיימה עם פלט y, $M_\emptyset$ מקבלת אם $y = 0$ ודוחה אחרת.

על פי הגדרה, $L(M) = \emptyset$ אם ורק אם $f(\langle M\rangle) = 0$ ומכאן נכונות המכונה $M_\emptyset$.

נעבור כעת להוכחה עקיפה כי f אינה ניתנת לחישוב. ראשית נתבונן על השפה L_f:

$$L_f = \{(\langle M\rangle, |L(M)|) \mid |L(M)| < \infty\}$$

אם נראה ש־$L_f \notin \mathrm{RE}$ זה יוכיח שהפונקציה אינה ניתנת לחישוב. נעשה זאת באמצעות רדוקציה $L_\emptyset \le L_f$ שתוגדר על ידי

$$\langle M\rangle \mapsto (\langle M\rangle, 0)$$

בבירור הרדוקציה תקפה, שכן $\langle M\rangle \in L_\emptyset$ אם ורק אם $L(M) = 0$. זה מסיים את ההוכחה במקרה זה.

דוגמא: פונקציית העצירה נתבונן כעת על פונקציה שמזכירה בהגדרתה את HP:

$$f(\langle M\rangle, x) = \begin{cases} 1 & M \text{ halts on } x \\ 0 & \text{else} \end{cases}$$

במקרה זה,

$$L_f = \{((\langle M\rangle, x), 1) \mid (\langle M\rangle, x) \in \mathrm{HP}\} \cup \{((\langle M\rangle, x), 0) \mid (\langle M\rangle, x) \notin \mathrm{HP}\}$$

והרדוקציה $\mathrm{HP} \le L_f$ הנתונה על ידי $w \mapsto (w,1)$ מוכיחה שאינה ניתנת לחישוב, שכן אם הייתה ניתנת לחישוב, עקב כך שהיא מלאה היה מתקיים $L_f \in \mathrm{R}$.

אם לעומת זאת היינו מרשים לפונקציה להיות לא מלאה:

$$g(\langle M\rangle, x) = \begin{cases} 1 & M \text{ halts on } x \\ \bot & \text{else} \end{cases}$$

אז במקרה זה היא הייתה ניתנת לחישוב ־ מכונה לחישוב g פשוט מריצה את M על x ומחזירה 1 אם הריצה הסתיימה.
אם היינו מחליפים את המקרה שבו הפונקציה לא מחזירה פלט:

$$h(\langle M\rangle, x) = \begin{cases} \bot & M \text{ halts on } x \\ 0 & \text{else} \end{cases}$$

אז היינו מקבלים $L_h = \left\{((\langle M\rangle, x), 0) \mid (\langle M\rangle, x) \in \overline{\mathrm{HP}}\right\}$ והרדוקציה $\overline{\mathrm{HP}} \le L_h$ הנתונה על ידי $w \mapsto (w, 0)$ הייתה מראה ש־$L_h \notin \mathrm{RE}$ ולכן h אינה ניתנת לחישוב (גם כשלוקחים בחשבון את העובדה שאינה מלאה).

3.6 בעיות זיהוי וחיפוש של יחסים

נעבור כעת לסוג נוסף של בעיות, שעומד להפוך למרכזי מאוד בחלקו השני של הקורס ־ בעיות של **זיהוי** ו**חיפוש** של יחסים.

כזכור, **יחס** הוא תת־קבוצה של זוגות של מילים, $S \subseteq \Sigma^* \times \Sigma^*$ (קיימות הגדרות כלליות יותר ליחס אך לא נזדקק להן). כל מה שנציג כעת תקף באופן כללי ליחסים כלשהם, אך לטובת האינטואיציה כדאי לחשוב על היחס S כמכיל זוגות (x, y) של איברים כך ש־x מתאר אובייקט מתמטי כלשהו ו־y מתאר פריט מידע מעניין כלשהו עליו. למשל, $S = \{(\langle M\rangle, w) \mid w \in L(M)\}$ הוא יחס שבו x מייצג מכונת טיורינג ו־y מייצג מילה שאותה היא מקבלת. דוגמא נוספת, מיקום מתמטי אחר, היא אוסף הזוגות (G, T) כך ש־G הוא גרף ו־T הוא עץ פורש של T.

אנו מבדילים בין שני סוגים של בעיות הקשורות ליחסים:

- **בעיית הזיהוי** היא הבעיה של להכריע האם זוג (x, y) קונקרטי שייך ליחס ־ האם M מקבלת את w? האם T הוא עץ פורש של G?
- **בעיית החיפוש** היא הבעיה שבה, בהינתן x, עלינו למצוא y כך ש־(x, y) שייך ליחס (אם קיים y כזה). האם בהינתן M אנו מסוגלים למצוא מילה שהיא מקבלת? האם בהינתן גרף G אנו יודעים למצוא לו עץ פורש?

נגדיר זאת פורמלית.

הגדרה 26.3 יהא יחס $S \subseteq \Sigma^* \times \Sigma^*$.

- אומרים ש**בעיית הזיהוי** של S ניתנת לפתרון אם $S \in \mathrm{RE}$.
- אומרים ש**בעיית החיפוש** של S ניתנת לפתרון אם קיימת מ"ט M_S כך שלכל $x \in \Sigma^*$, אם קיים y כך ש־$(x, y) \in S$ אז M_S עוצרת על x עם פלט y' כך ש־$(x, y') \in S$ (לאו דווקא $y = y'$) ואילו אם לא קיים y כזה אז M_S אינה עוצרת.

נשים לב לכך שעבור פונקציה f, השפה $L_f = \{(x, y) \mid f(x) = y\}$ היא עצמה יחס. בעיית הזיהוי של יחס זה היא הבעיה של בדיקה האם f מוגדרת על x והפלט שלה שווה ל־y, ובעיית החיפוש היא הבעיה של חישוב f על x. עם זאת, זהו מקרה פרטי שכן באופן כללי עבור יחסים ל־x יכול להיות יותר מ־y אחד כך ש־(x, y) ביחס.

משפט 27.3 אם S ניתן לזיהוי, אז S ניתן לחיפוש.

הוכחה: בהינתן x נבצע הרצה מבוקרת על כל ה־y האפשריים. לכל (x, y) נריץ על הזוג את המכונה שמזהה את S. אם אחת מההרצות הסתיימה בקבלה של הזוג (x, y), נוציא את y כפלט. ■

משפט זה היה פשוט מאוד להוכחה, בהינתן ההיכרות שלנו עם הרצה מבוקרת; הטענה המקבילה בחלקו השני של הקורס תהיה **השאלה הפתוחה המרכזית של מדעי המחשב התיאורטיים**. אינטואטיבית, הסיבה להבדל נעוצה בכך שבחלק השני נדבר על זיהוי וחיפוש **יעילים** מבחינת זמן ריצה, אבל הרצה מבוקרת היא טכניקה **לא יעילה** מבחינת זמן ריצה.

הכיוון השני של המשפט כלל אינו נכון. יחס S יכול להיות ניתן לחיפוש למרות שאינו ניתן לזיהוי: למשל, אם נחשוב על השפה L_{EQ} בתור יחס, היא אינה ניתנת לזיהוי כי ראינו כבר ש־$L_{\mathrm{EQ}} \notin \mathrm{RE}$. מצד שני, בעיית החיפוש שלה פתירה בצורה "טיפשית": בהינתן $\langle M\rangle$, הפלט שלנו יהיה $\langle M\rangle$, שכן $(\langle M\rangle, \langle M\rangle) \in L_{\mathrm{EQ}}$ שהרי כל מכונה שקולה לעצמה.

דוגמא: השפה L_u נתבונן על השפה $L_u = \{(\langle M \rangle, x) \mid x \in L(M)\}$ בתור יחס.

- **בעיית הזיהוי** של השפה ניתנת לפתרון, מכיוון ש־$L_u \in \mathrm{RE}$ (בהינתן $(\langle M \rangle, x)$) מריצים את M על x ובודקים אם הריצה הסתיימה בקבלה).
- **בעיית החיפוש** של השפה ניתנת לפתרון, שכן ראינו כי יחס ניתן לזיהוי הוא יחס ניתן לחיפוש.

נתבונן כעת על השפה המשלימה $\overline{L_u} = \{(\langle M \rangle, x) \mid x \notin L(M)\}$ בתור יחס.

- **בעיית הזיהוי** של השפה אינה ניתנת לפתרון, שכן $\overline{L_u} \notin \mathrm{RE}$ כפי שראינו קודם.
- **בעיית החיפוש** של השפה אינה ניתנת לפתרון. כדי לראות זאת, נשים לב לכך שאם בעיית החיפוש הייתה ניתנת לפתרון, זה היה מניב מכונה שמקבלת את השפה $\overline{L_{\Sigma^*}} = \{\langle M \rangle \mid L(M) \neq \Sigma^*\}$; בהינתן $\langle M \rangle$, היינו מחפשים x כך ש־$x \notin L(M)$. מכאן ש־$\overline{L_{\Sigma^*}} \in \mathrm{RE}$. מצד שני, משפט רייס מראה מייד כי $\overline{L_{\Sigma^*}} \notin \mathrm{RE}$.

3.7 סיבוכיות קולמוגורוב

נעסוק כעת בבעיית חישוב פונקציה שאינה פתירה, וניתן להראות זאת באופן "ישיר", שאינו עובר דרך שימוש בבעיות הלא כריעות שכבר ראינו ־ הבעיה של חישוב **סיבוכיות קולמוגורוב** של מחרוזת.

המטרה של סיבוכיות קולמוגורב היא לתת מדד כמותי ל"אקראיות" של מחרוזת ־ ככל שמחרוזת היא פחות תבניתית ויותר אקראית למראה, הסיבוכיות שלה אמורה לעלות. כך למשל עבור שלוש המחרוזות

- 000000000000000000
- 010101010101010101
- 011010110010100101

המחרוזת הראשונה פשוטה מאוד, השניה רק מעט יותר מורכבת, והשלישית כבר "אקראית" ללא תבנית ברורה.

הגדרה 28.3 סיבוכיות קולמוגורוב של מחרוזת $x \in \Sigma^*$ המסומנת $k(x)$ היא מספר המצבים הקטן ביותר של מ"ט M עם $\Sigma = \{0,1\}, \Gamma = \{0,1,\flat\}$ כך ש־M על ε פולטת x.

נשים לב לכך ש־k היא פונקציה מלאה: כל מחרוזת x ניתנת לייצור על ידי מכונת טיורינג ש־x הוא חלק מהקידוד שלה. ראינו מכונות רבות כאלו עד כה אבל לצורך השלמות נראה מכונה כזו במפורש.

באופן כללי ניתן לכתוב $x = x_1 x_2 \ldots x_m$ כאשר $x_i \in \Sigma$ לכל $1 \le i \le m$. נבנה מכונה שמצביה הן סיפות של x, כלומר $Q = \{q_{x_i \ldots x_m} \mid 1 \le i \le m\} \cup \{q_\varepsilon\}$ כאשר $F = \{q_\varepsilon\}$.

פונקציית המעברים שלנו תאמר "כתוב את התו הבא שאתה אמור לכתוב, כלומר את התו השמאלי ביותר בסיפא שאתה מחזיק כרגע, זוז צעד אחד שמאלה והסר את התו שכתבת מהסיפא". פורמלית, $\delta(q_{x_i \ldots x_m}, \sigma) = (q_{x_{i+1} \ldots x_m}, x_i, R)$. כאן אנו משתמשים בקונבנציה ש־$x_{m+1} \ldots x_m$ הוא פשוט המחרוזת הריקה ε.

קל לראות שהמכונה שתיארנו אכן מוציאה את x כפלט על ε, ויש לה בדיוק $m+1$ מצבים כך שגם הראינו ש־$k(x)$ מוגדרת וגם ש־$k(x) \le m+1$, כלומר לכל $x \in \Sigma^*$ מתקיים $k(x) \le |x| + 1$.

מטרתנו היא להוכיח כי $k(x)$ אינה ניתנת לחישוב. אם היינו יכולים להניח כי מכונת טיורינג M שאנו בונים יודעת את $\langle M \rangle$, ההוכחה הייתה פשוטה למדי: M הייתה עוברת סדרתית על כל ה־$x \in \Sigma^*$ ומחשבת את $k(x)$ לכל אחד מהם. בהמשך נראה כי k היא פונקציה לא חסומה, כך שמתישהו M הייתה מוצאת x כך ש־$k(x) > |Q_M|$; בשלב זה M הייתה עוצרת ומוציאה את x כפלט, וכך היינו מגיעים לסתירה ־ כי סיבוכיות הקולמוגורב של x היא $k(x)$ אבל M היא מכונה עם **פחות** מצבים מ־$k(x)$ שפולטת את x.

בפועל אנחנו באמת יכולים להניח כי M יודעת את $\langle M \rangle$; זהו תוכן **משפט הרקורסיה של קלייני** שהזכרנו מוקדם יותר בקורס. אולם לא נניח כאן כי יש לנו אותו, ולכן ננקוט ב"תעלול" טכני שמאפשר לנו להשיג אפקט דומה.

נתחיל עם הטענה הקריטית לנו ־ כי k היא פונקציה לא חסומה, כך שמעבר סדרתי על כל ה־x־ים וחישוב k עבורם בהכרח יניב מספרים גדולים כרצוננו:

טענה 29.3 לכל n טבעי קיים $x \in \Sigma^*$ כך ש־$k(x) \ge n$.

הוכחה: קיים רק מספר סופי של מכונות טיורינג לא שקולות זו לזו עם $|Q| < n$, שכן מספר מכונות הטיורינג שאינן שקולות חסום על ידי מספר הקידודים של מכונות טיורינג, וכאשר $|Q|,|\Gamma|$ חסומים גם גודל הקידוד חסום. מכיוון שכל מכונת טיורינג מייצרת מחרוזת בודדת על ε, קיים רק מספר סופי של מחרוזות המיוצרות על ידי מ"ט עם $|Q| < n$. מכיוון ש־Σ^* אינסופית, קיימת בה מחרוזת x שאינה שייכת לקבוצה הסופית של המחרוזות המיוצרות על ידי מ"ט עם $|Q| < n$. ■

נעבור כעת להוכחת הטענה המרכזית. במקום לבנות מ"ט בודדת שמייצרת מחרוזת שהיא "מורכבת מדי מכדי שהמכונה תוכל לייצר אותה" (מה שדרש מהמכונה להכיר את הקידוד של עצמה), נבנה **סדרה** של מכונות, כך שמובטח לנו שאם נתקדם מספיק בסדרה נגיע אל מכונה שמייצרת מחרוזת "מורכבת מדי".

משפט 30.3 הפונקציה $k(x)$ אינה ניתנת לחישוב.

הוכחה: נניח ש־k ניתנת לחישוב בעזרת מ"ט K ונגיע לסתירה.
לכל n טבעי, נבנה מ"ט M_n שפועלת כך על כל קלט:

- כותבת על הסרט את המספר n בכתיב בינארי.
- רצה בהרצה מבוקרת על כל ה־$x \in \Sigma^*$ ומחשבת את $k(x)$ לכל x כזה על ידי הרצת K.
- אם התגלה x כך ש־$K(x) \geq n$, המכונה M_n עוצרת עם פלט x.

מספר המצבים של M_n מורכב משלושה רכיבים:

- הרכיב שכותב n על הסרט. מכיוון ש־n בכתיב בינארי הוא מאורך $\lg n$ אז נדרשים $O(\lg n)$ מצבים לצורך כך.
- הרכיב שכולל את K. מספר המצבים של K אינו תלוי ב־n כך שהוא כולל $O(1)$ מצבים.
- הרכיב שאחראי לביצוע הרצת K וההרצה המבוקרת על $x \in \Sigma^*$. גם רכיב זה כולל $O(1)$ מצבים.

בסך הכל, מספר מצבי M_n הוא $O(\lg n)$ ולכן הוא $o(n)$. כלומר, קיים n_0 כך שאם $n > n_0$ אז $|Q_{M_n}| < n$. מצד שני, על פי הגדרתה, M_n מוציאה כפלט x כך ש־$k(x) \geq n$ בתנאי שקיים כזה. המשפט הקודם שהוכחנו הראה שתמיד קיים כזה, כך ש־M_n תוציא כפלט x כך ש־$k(x) \geq n > |Q_{M_n}|$ והגענו לסתירה המבוקשת. ■

4 מבוא לתורת הסיבוכיות

4.1 הגדרת חישוב יעיל

עד כה כל הבעיות החישוביות שעסקנו בהן הניחו שהמשאבים שעומדים לרשות מכונת הטיורינג שפותרת אותן הם **בלתי מוגבלים**. כעת נעבור לעסוק בשאלה מה קורה כאשר אנו מגבילים חלק ממשאבים אלו. יש סוגים רבים של משאבים שניתן לעסוק בהם, אבל שני הנפוצים ביותר בתיאוריה של מדעי המחשב הם **זמן החישוב** ו**הזיכרון שנדרש לצורך החישוב**.

אחת מהבעיות שבהן נתקלים כשמעוניינים למדוד את צריכת המשאבים הללו של אלגוריתם מסויים היא שלא ברור כיצד למדוד אותם ־ האם זמן חישוב נמדד בשניות? אבל אם כן, כיצד ניתן לחשב את זמן החישוב עבור אלגוריתם נתון? האם עלינו לקודד ולהריץ אותו על מחשב מסוים? אבל במחשבים שונים האלגוריתם ירוץ זמנים שונים בשל יעילות המעבד, אופטימיזציות שמתבצעות ברמת המעבד, אופטימיזציות בזמן הקומפליציה וכיוצא בזה. אפילו תנאים חיצוניים כמו החום בסביבת המעבד עשויים להשפיע על זמן הריצה. מכאן הרצון למצוא הגדרה **תיאורטית** של זמן ריצה, שאינה תלויה בחומרה זו או אחרת. מכונת טיורינג היא סביבה טבעית להגדרה כזו:

הגדרה 1.4 זמן הריצה של מכונת טיורינג M על קלט x הוא מספר צעדי החישוב ש־M מבצעת על x.

נשים לב לכך שמספר צעדי החישוב עשוי להיות אינסופי, אבל כמובן שסיטואציה כזו אינה רלוונטית לנו אם אנו מעוניינים להגביל את זמן הריצה.

בעיה נוספת היא שעל פי רוב, זמן הריצה של אלגוריתם תלוי **בגודל הקלט** שמוזן אליו. נתבונן על מ"ט פשוטה במיוחד ־ כזה שמכריע את השפה $L = \{1^n \mid n \in \mathbb{N}\}$ של כל המחרוזות האונריות. על קלט x, המכונה פשוט עוברת סדרתית על כל תווי x. אם אחד מהם הוא 0 היא דוחה, ואם הגיעה אל ה־$\flat$ שבסוף הקלט היא מקבלת. מה מספר צעדי החישוב של המכונה על קלט? היא מבצעת לכל היותר $|x| + 1$ צעדים; מספר זה תלוי באורך הקלט. ככל שהקלט ארוך יותר, כך המכונה תבצע צעדי חישוב רבים יותר. באופן כללי, אם מכונה על קלט x מבצעת פחות מ־$|x|$ צעדי חישוב, המשמעות היא שהמכונה כלל לא קראה את כל הקלט, וזה אינו מקרה נפוץ במיוחד (אם כי הוא בהחלט קיים). אם כן, ברור שמדידת זמן הריצה שלנו היא תמיד **ביחס לאורך הקלט**.

הגדרה 2.4 תהא $f:\mathbb{N}\to\mathbb{N}$ פונקציה ותהא M מ"ט. אומרים ש־M פועלת **בסיבוכיות זמן ריצה** $O\left(f\right)$ אם לכל x, זמן ריצת M על x הוא $O\left(f\left(|x|\right)\right)$.

נקודה עדינה שנובעת מהגדרת סיבוכיות זמן ריצה בתור פונקציה של אורך הקלט היא שיש חשיבות **לאופן הייצוג** של הקלט כשאנו מנסים לקבוע אם אלגוריתם הוא יעיל או לא. דוגמא קלאסית היא האלגוריתם הנאיבי לבדיקת ראשוניות: עבור מספר n, ניתן לבדוק אם n ראשוני על ידי מעבר סדרתי על כל המספרים $1<k<n$ ובדיקה האם k מחלק את n. אם כן ־ דוחים, ואם לכל k הבדיקה נכשלה, מקבלים. אלגוריתם זה מבצע $O\left(n\right)$ פעולות חלוקה ולכן הוא לכאורה יעיל, כי "זמן ריצה לינארי הוא יעיל", אך בפועל זה אלגוריתם לא יעיל מאוד. הסיבה לכך היא שכדי לייצג את המספר n אנו זקוקים רק ל־$\lg n$ ביטים, והאלגוריתמים שלנו לחיבור, כפל וכו' של מספרים פועלים בסיבוכיות $O\left(\lg n\right)$ ו־$O\left(\lg^2 n\right)$ וכדומה. כלומר, אם n מיוצג בבסיס בינארי אז האלגוריתם שראינו דורש זמן אקספוננציאלי **בגודל הייצוג** של n. לעומת זאת, אם n מיוצג בבסיס אונרי, כלומר בתור 1^n, אז האלגוריתם לבדיקת ראשוניות אכן יהיה $O\left(n\right)$.

בעיה נוספת עם הגדרת סיבוכיות זמן ריצה היא שגם במודל האבסטרקטי של מכונת טיורינג, אנחנו עדיין עלולים לבצע הזנחות בעייתיות. למשל, בהצגה של אלגוריתם בדיקת הראשוניות אמרנו שהוא מבצע $O\left(n\right)$ "פעולות חלוקה". אולם פעולת חלוקה בעצמה אינה פעולה אטומית, אלא היא דורשת פירוק לתת־פעולות, ויש למנות גם את תת־הפעולות הללו. בפועל, ברוב המקרים שבהם מנתחים סיבוכיות של אלגוריתמים הסיבוכיות נמדדת במספר "פעולות בסיס" שהאלגוריתם מבצע, כאשר חלוקה יכולה להיחשב לפעולת בסיס שכזו; אבל בהגדרה שלנו פעולת הבסיס היחידה היא **צעד** של מכונת טיורינג.

דוגמא קלאסית לאופן שבו ניסוח מילולי של פעולת מכונת טיורינג מסתיר תת־פעולות שאינן אטומיות הוא "המכונה M תריץ את המכונה M' ותענה כמוה". כזכור, באופן שבו ביצענו הרצה מבוקרת נזקקנו למכונה רב־סרטית M שבכל צעד חישוב שלה סורקת את כל פונקציית המעברים של M'. סריקה שכזו גורמת לניפוח בזמן הריצה, שכן בריצה של מכונת טיורינג רגילה, פונקציית המעברים "מופעלת אוטומטית" בלי שיתבצעו צעדי חישוב כלשהם.

גם עצם השימוש במכונה רב־סרטית מסתיר חיסכון בזמן ריצה; האופן שבו מכונה חד־סרטית מסמלצת ריצת מכונה רב־סרטית יוצר גם הוא גידול בזמן הריצה. פירוש הדבר הוא שהגדרת זמן הריצה שנתנו היא **תלויה במודל** של מכונת הטיורינג שאיתו אנחנו עובדים. טבעי לבחור בתור מודל את המכונה הפשוטה ביותר, החד־סרטית; אבל אז, כאשר אנו מתארים אלגוריתמים בצורה מילולית, זמן הריצה שנתאר לא יהיה מדויק.

ניתן להתמודד עם קשיים אלו, אך לא נעשה זאת בקורס, זאת מכיוון שההגדרה של זמן חישוב **יעיל** שבה נשתמש מאפשרת להתעלם מכל ההבדלים הללו. לפני שנציג הגדרה זו ניתן מוטיבציה אחת נוספת לשימוש בה.

כזכור, רדוקציה $L_1\le L_2$ מאפשרת לנו להכריע את השפה L_1 אם בידינו אלגוריתם שמכריע את L_2. היינו רוצים להשתמש ברדוקציות גם בהקשר של סיבוכיות זמן ריצה, באופן הבא: אם בידינו אלגוריתם יעיל להכרעת L_2 ואנו יודעים לחשב ביעילות את הרדוקציה מ־L_1 אל L_2, אז גם L_1 ניתנת לפתרון ביעילות. כעת, אם קיבלנו קלט x שעלינו להכריע אם הוא שייך ל־L_1, אחרי הפעלת הרדוקציה נקבל פלט $y=f\left(x\right)$. אורכו של y עשוי להיות גדול מאורכו של x. זמן הריצה של האלגוריתם היעיל להכרעת L_2 נמדד ביחס לאורך של y, לא של x. מכך נובע שהמושג שלנו של "זמן ריצה יעיל" צריך להיות סגור ל**הרכבה**: אם a,b הן שתי פונקציות שמתארות זמן ריצה יעיל, גם $b\circ a$ צריכה לתאר זמן ריצה יעיל, שכן זה עשוי להיות זמן הריצה שנדרש כדי להכריע את L_1.

למשל, אם נקבע ש"זמן ריצה יעיל" הוא זמן ריצה $O\left(n^2\right)$, אז אחרי הפעלת f אנו עלולים לקבל פלט y כך ש־$|y|=|x|^2$, ואז הכרעת L_2 עלולה לדרוש זמן ריצה $O\left(|y|^2\right)=O\left(|x|^4\right)$, ואנו נדחפים לכך שגם $O\left(n^4\right)$ ייחשב זמן ריצה יעיל, ובצורה זו אנו מגיעים לכך ש־$O\left(n^c\right)$ ייחשב זמן ריצה יעיל לכל $c\in\mathbb{N}$. זו אכן ההגדרה שבה נשתמש:

הגדרה 3.4 מכונת טיורינג M תיקרא **פולינומית** או **יעילה** אם קיים $c\in\mathbb{N}$ כך ש־M פועלת בסיבוכיות זמן ריצה $O\left(n^c\right)$.

מכאן ואילך נתבסס בצורה אינטנסיבית על הגדרה זו ל"יעילות" ולכן מוטב לזכור את הסיגים המתבקשים אליה:

- לא כל אלגוריתם "יעיל" ייתפס על ידינו כיעיל בפועל. אלגוריתם עם סיבוכיות זמן ריצה של $O\left(n^{100}\right)$ בהחלט עשוי להיות בלתי פרקטי בעליל לכל צורך מעשי. כדי להבין מדוע זה עדיין מתקבל על הדעת, נזכור שמטרתנו בהמשך תהיה להוכיח שבעיות מסויימות הן (בסבירות גדולה) **לא יעילות**, כלומר ש**אפילו** אלגוריתם בסיבוכיות $O\left(n^{100}\right)$ הוא יותר ממה שניתן לקוות לו עבורן.

- בהחלט **קיימים** אלגוריתמים שאנחנו משתמשים בהם בפועל שפועלים בסיבוכיות ריצה שאינה פולינומית. דוגמא מפורסמת היא **אלגוריתם הסימפלקס** לפתרון בעיות תכנון לינארי, אך נזכיר מקרים נוספים בהמשך.

- בהמשך לנקודה הקודמת, האופן שבו אנו מודדים זמן ריצה הוא ביחס **למקרה הגרוע ביותר**. בפועל, המקרה הגרוע ביותר אינו בהכרח מעיד על המקרה הממוצע, או המקרה הנפוץ בפועל; אבל מושג לא מוגדר היטב כמו "המקרה הנפוץ בפועל" מונע מאיתנו ניתוח תיאורטי מלכתחילה, כך שאנו מתמקדים בדברים שאנו כן יכולים לדבר עליהם ונזהרים לא לייחס להם משמעות גדולה יותר מדי.

הגדרה 4.4 (מחלקות חישוב יעיל)

- המחלקה P היא אוסף השפות שקיימת מכונת טיורינג פולינומית M המקבלת אותן.
- המחלקה POLY היא אוסף הפונקציות שקיימת מכונת טיורינג פולינומית M המחשבת אותן.

נשים לב שחסם על זמן ריצה של מכונה גורר אוטומטית את העצירה שלה על כל קלט, ולכן:

טענה 5.4 $\mathrm{P}\subseteq\mathrm{R}$ ואם $f\in\mathrm{POLY}$ אז f מלאה.

הוכחה: תהא $L\in\mathrm{P}$ ותהא M מ"ט פולינומית כך ש־$L\left(M\right)=L$. מכיוון ש־M פולינומית, היא עוצרת על כל קלט ולכן M מכריעה את L ו־$L\in\mathrm{R}$.
תהא $f\in\mathrm{POLY}$ ותהא M מ"ט פולינומית כך ש־$f_M=f$. מכיוון ש־M פולינומית, היא עוצרת על כל קלט ולכן f מוגדרת על כל קלט. ■

עם זאת, נשים לב לכך שהטענה $L\in\mathrm{P}$ אין משמעותה ש**כל** מ"ט M כך ש־$L\left(M\right)=L$ מכריעה את L. למשל, עבור $L=\emptyset$ המכונה שדוחה מייד כל קלט היא מ"ט פולינומית עבור L, אך גם המכונה שאינה עוצרת לכל קלט.
אילו בעיות שאנו מכירים שייכות ל־P? למשל, רוב הבעיות שנלמדות בקורס סטנדטי של אלגוריתמים בתורת הגרפים:

דוגמא יהא $G=\left(V,E\right)$ גרף. בקורס זה נניח כי גרפים מיוצגים על ידי רשימה מפורשת של אברי V ואוסף הקשתות E, כך ש־$\left|V\right|,\left|E\right|=O\left(\left|G\right|\right)$ (זאת להבדיל מייצוגים מובלעים של גרפים שבהם גודל הייצוג של הגרף עשוי להיות קטן משמעותית ממספר צמתיו/קשתותיו; נציג דוגמא לסיטואציה כזו בהמשך הקורס).
תחת הנחת ייצוג זו, בעיות ההכרעה הבאות שייכות ל־P:

- האם G קשיר או לא.
- האם קיים מסלול ב־G בין הצמתים a,b.
- בהינתן פונקציית משקל על קשתות G, האם קיים מסלול ב־G בין הצמתים a,b ממשקל קטן מערך נתון.
- בהינתן פונקציית משקל על קשתות G, האם קיים ל־G עץ פורש ממשקל קטן מערך נתון.
- בהינתן רשת זרימה $\left(G,s,t,c\right)$, האם קיימת זרימה ברשת מערך גדול או שווה לערך נתון.

דוגמא תהא L שפה סופית כלשהי, אז $L\in\mathrm{P}$. כדי לראות זאת, נבנה מ"ט M שבודקת אם הקלט x שייך לרשימת המילים בשפה L. מכיוון שאורך המילים בשפה הוא חסום, המכונה לא צריכה לקרוא x־ים שארוכים מהמילים בשפה ויכולה פשוט לדחות; מכאן שזמן הריצה שלה הוא $O\left(1\right)$ ואינו תלוי בגודל של x; זו דוגמא לסיטואציה שבה המכונה אינה צריכה לקרוא את כל הקלט שלה.
נתאר פורמלית את הבניה כדי להשתכנע שאנחנו לא מחביאים פרטי מימוש. ראשית יהא n האורך המקסימלי של מילה ב־L (קיים אורך מקסימלי כי השפה סופית). כעת נבנה את M באופן הבא: קבוצת המצבים תהיה $Q=\left\{q_w\mid\left|w\right|\le n\right\}\cup\left\{q_{acc},q_{rej}\right\}$, כלומר יש לנו מצב לכל מילה מאורך עד n.
הרעיון הוא שהמצב שבו אנחנו נמצאים אומר לנו מה הרישא של הקלט x שקראנו עד כה. בהתאם לכך, המצב ההתחלתי יהיה q_ε, ופונקציית המעברים תהיה $\delta\left(q_w,\sigma\right)=\left(q_{w\sigma},\sigma,R\right)$ לכל $\sigma\ne\flat$ ו־w כך ש־$\left|w\right|<n$.
אם המכונה קראה כבר n תווים אבל לא הגיעה לסוף הקלט, אנחנו יודעים שהקלט ארוך מדי להיות שייך לשפה ולכן עלינו לדחות. את זה משיגים על ידי המעברים $\delta\left(q_w,\sigma\right)=\left(q_{rej},\sigma,S\right)$ לכל $\sigma\ne\flat$ ו־w כך ש־$\left|w\right|=n$.
בנוסף, אם המכונה הגיעה אל התו $\flat$ היא סיימה את קריאת הקלט, ולכן ניתן לבדוק אם הוא שייך לשפה או לא. זה מושג על ידי המעבר

$$\delta\left(q_w,\flat\right)=\begin{cases}\left(q_{acc},\flat,S\right) & w\in L\\ \left(q_{rej},\flat,S\right) & w\notin L\end{cases}$$

הסיבה שבגללה לא ניתן להשתמש בבניה הזו כדי להכריע שפות אינסופיות היא שבמקרה זה, נזדקק לאינסוף מצבים של המכונה (ולכן, אם היינו מגדירים מכונת טיורינג כך שקבוצת המצבים שלה יכולה להיות אינסופית היינו יכולים להכריע כל שפה באופן טריוויאלי).

דוגמא השפה $\mathrm{PRIMES}=\left\{n\in\mathbb{N}\mid n\text{ is prime}\right\}$ שייכת ל־P. תוצאה זו הייתה במשך שנים רבות בעיה פתוחה, עד המאמר "PRIMES in P" משנת 2002 של Agrawal, Kayal, Saxena (האלגוריתם במאמר נקרא אלגוריתם AKS על שמם).

4.2 המחלקה NP

4.2.1 מבוא

סודוקו הוא משחק לשחקן יחיד שבו לוח 9×9 משבצות, והמטרה היא למלא את כל משבצות הלוח במספרים בין 1 ל־9, כך שבכל שורה ועמודה כל המספרים יהיו שונים זה מזה, מה שמכונה במתמטיקה **ריבוע לטיני**. בנוסף לכך בסודוקו קיים עוד אילוץ: מחלקים את הלוח ל־9 תתי־ריבועים של 3×3 והדרישה היא שגם בכל אחד מהם כל המספרים יהיו שונים זה מזה.

קל לבנות לוח סודוקו שעומד בכל הדרישות הללו, ולכן משחק סודוקו כולל אתגר נוסף: **חלק** ממשבצות הלוח כבר מכילות מספרים, ויש **להשלים** את הלוח תוך שמירה על מספרים אלו. בחירה שונה של מצב התחלתי של הלוח מניבה חידת סודוקו אחרת.

פתרון חידת סודוקו כולל הפעלה של כללי אלימינציה פשוטים (למשל, אם המשבצת שלנו נמצאת בשורה שבה כבר מופיע 4, אסור למשבצת שלנו להכיל 4) אבל בפתרון חידות סודוקו מדי פעם נקלעים למצב שבו חייבים "לנחש": לכתוב מספר באחת המשבצות, לבדוק אם אפשר לפתור את הלוח בסיוע מהלך זה, ואם הגענו למבוי סתום ־ למחוק את המספר שכתבנו ולנסות אחר. ברור שניחוש לבדו לא יאפשר לנו לפתור את חידת הסודוקו ־ יש 9 אפשרויות לכל משבצת, כך שניחוש של n משבצות מוביל ל־9^n אפשרויות שונות שחייבים לבדוק, וזהו מספר אקספוננציאלי שהופך לבלתי סביר מהר מאוד; אלימינציה דטרמיניסטית כלשהי היא הכרחית.

אם כן, הקושי של משחק סודקו אינו לגמרי ברור ונראה שהוא נעוץ במספר הפעמים שבהן ניאלץ "לנחש", אבל בסודוקו כן יש אספקט שבו הוא פשוט מאוד: אם בעיית סודוקו כבר נפתרה, **קל** לנו לבדוק שהפתרון הוא אכן לגיטימי ולא כולל "רמאויות". כל שאנו נדרשים לעשות הוא לבדוק את הלוח שהוא הפתרון: להסתכל על כל שורה, עמודה ותת־ריבוע ולוודא שכל המספרים בהם שונים זה מזה, ולוודא שכל משבצת שכללה מספר בחידה המקורית כוללת את אותו מספר גם בפתרון. בדיקות אלו מתבצעות בצורה ישירה ובזמן קצר.

משחק הסודוקו הוא דוגמא טיפוסית למה שנכנה בהמשך "בעיית NP" ־ בעיה חישובית שעשויה להיות קשה לפתרון, אבל קל לבדוק לה פתרון. בעיות אלו צצות בכל תחומי מדעי המחשב, לעתים קרובות בהקשרים פרקטיים מאוד, ולכן השאלה האם **לכל** בעיה ב־NP קיים פתרון יעיל היא מעניינת; כל כך מעניינת, עד שהיא זכתה למעמד של הבעיה הפתוחה המרכזית של מדעי המחשב התיאורטיים ־ בעיית P=NP.

4.2.2 הגדרה פורמלית

כזכור, עסקנו קודם בבעיות **זיהוי וחיפוש** של יחסים. נרצה לחזור להגדרות אלו גם בהקשר של סיבוכיות זמן יעילה. האינטואיציה שלנו היא שיחס $S \subseteq \Sigma^* \times \Sigma^*$ מתאר זוגות (x,y) של "בעיה" ו"פתרון" עבורה. למשל, x יכול לקודד לוח סודוקו, ו־y הוא פתרון מוצע עבורו. בעיית הזיהוי האם $(x,y) \in S$ היא הבעיה של **בדיקת פתרון** ובעיית החיפוש שבה בהינתן x יש למצוא y כך ש־$(x,y) \in S$ היא הבעיה של **מציאת פתרון**.

הגדרה 6.4 יחס S הוא **ניתן לזיהוי פולינומי** (או **ניתן לזיהוי יעיל**) אם $S \in \mathrm{P}$.

כשהגדרנו את בעיית החיפוש בהקשר של מכונות טיורינג שאינן מוגבלות בזמן, הרשנו למכונה לא לעצור אם אין לה פלט מתאים להוציא. בהקשר של מכונות עם חסמי זמן ריצה עלינו להשתמש בהגדרה עדינה יותר:

הגדרה 7.4 יחס S הוא **ניתן לחיפוש פולינומי** (או **ניתן לחיפוש יעיל**) אם קיימת מ"ט פולינומית M עם מצבים סופיים $F = \{q_{acc}, q_{rej}\}$ כך שלכל $x \in \Sigma^*$:

- אם לא קיים y כך ש־$(x,y) \in S$ אז M עוצרת במצב q_{rej}.
- אחרת, המכונה עוצרת במצב q_{acc} עם פלט y כלשהו כך ש־$(x,y) \in S$.

עם זאת, קיימת נקודה עדינה נוספת שעלינו להתייחס אליה. מ"ט פולינומית M שבודקת האם $(x,y) \in S$ צריכה להיות פולינומית בגודל הקלט; במקרה זה, הקלט כולל הן את x והן את y, כך ש־M צריכה להיות פולינומית ב־$|x| + |y|$. זה פותח פתח לסיטואציה אבסורדית שבה x מייצג בעיה מורכבת כלשהי (נאמר, לוח סודוקו) אבל $y = 1^n$ הוא פשוט מחרוזת ארוכה מאוד של 1־ים. בהינתן y ארוך מאוד, מכונה M יכולה לפתור את x פשוט על ידי ביצוע חיפוש ממצה על כל האפשרויות, גם אם מספר האפשרויות הוא אקספוננציאלי ב־$|x|$. אנו מעוניינים למנוע סיטואציה כזו; אנו רוצים ש־y יסייע לנו בבדיקת x בזכות תוכן אינפורמטיבי שיש בו, לא בגלל שהוא מאפשר לנו לבצע מניפולציה של הגדרת המושג "יעיל". לכן נוסיף עוד דרישה:

הגדרה 8.4 יחס $S \subseteq \Sigma^* \times \Sigma^*$ הוא **חסום פולינומית** אם קיים פולינום p כך שלכל $(x,y) \in S$ מתקיים $|y| \le p(|x|)$.

אם יחס אינו חסום פולינומית, בהחלט ייתכן שהוא יהיה ניתן לזיהוי יעיל אבל לא לחיפוש יעיל, מהסיבה הטריוויאלית הבאה: אם $|y|$ הוא אקספוננציאלי ב־$|x|$, אז בהינתן קלט x, מ"ט פולינומית M לא תוכל לפלוט את y פשוט כי אין לה מספיק זמן לכתוב את כל y לסרט.

כאשר אנו מוסיפים את הדרישה ש־S יהיה חסום פולינומית אנו מנטרלים את המקרים הללו, ונשארים עם השאלה הבאה:

- בהינתן יחס חסום פולינומית S, האם העובדה ש־S ניתן לזיהוי יעיל תמיד גוררת ש־S ניתן לחיפוש יעיל?

זוהי **שאלה פתוחה** השקולה לשאלת $\mathrm{P}{=}\mathrm{NP}$ שנציג בקרוב; נוכיח את השקילות בהמשך.

נעבור כעת להגדרת המחלקה NP. זוהי מחלקה של **שפות**, כלומר של בעיות הכרעה. למשל, האם בהינתן לוח סודוקו בכלל קיים לו פתרון:

הגדרה 9.4 שפה L שייכת למחלקה NP אם קיים יחס R_L כך ש:

- R_L ניתן לזיהוי פולינומי.
- R_L חסום פולינומית.
- $L=\{x\in\Sigma^*\mid\exists y\in\Sigma^*:(x,y)\in R_L\}$

4.2.3 דוגמאות

מסלולים המילטוניים בהינתן גרף לא מכוון $G=(V,E)$, **מסלול המילטוני** בגרף הוא מסלול שעובר בכל צומת בגרף בדיוק פעם אחת ויחידה. נסמן ב־HL את שפת כל הגרפים שיש בהם מסלול המילטוני.

נגדיר יחס R_{HL} עבור השפה. היחס יכלול את כל הזוגות (G,p) של גרף G ומסלול המילטוני p בו. בבירור $\mathrm{HL}=\{G\mid\exists p:(G,p)\in R_{\mathrm{HL}}\}$.

נשים לב לכך שמסלול המילטוני הוא פשוט סדרה של אברי V מאורך $|V|$, כך שזהו יחס חסום פולינומית ב־$|G|$.

היחס R_{HL} ניתן לזיהוי פולינומי שכן בהינתן p מספיק מעבר אחד עליו כדי לקבוע את העובדות הבאות:

- כל $v\in V$ מופיע ב־p בדיוק פעם אחת (ניתן לתחזק מערך שמונה לכל $v\in V$ את מספר המופעים שלו ב־p).
- אם $p=v_1,v_2,\ldots,v_n$ אז לכל $1\le i<n$ מתקיים $(p_i,p_{i+1})\in E$.

שפות ב־P

טענה 10.4 אם $L\in\mathrm{P}$ אז $L\in\mathrm{NP}$, כלומר $\mathrm{P}\subseteq\mathrm{NP}$.

הוכחה: נתבונן ביחס $R_L=\{(x,\varepsilon)\mid x\in L\}$.

בבירור $L=\{x\in\Sigma^*\mid\exists y\in\Sigma^*:(x,y)\in R_L\}$.

היחס R_L חסום פולינומית כי אם $(x,y)\in R_L$ אז $|y|=|\varepsilon|=0=p(|x|)$ כאשר p הוא פולינום האפס.

היחס R_L ניתן לזיהוי פולינומי כי בהינתן (x,y), בדיקה ש־$(x,y)\in R_L$ כוללת בדיקה ש־$y=\varepsilon$ (טריוויאלי) ובדיקה ש־$x\in L$ (ניתן לביצוע בזמן פולינומי $L\in\mathrm{P}$). ■

סודוקו דוגמת הסודוקו שבה פתחנו היא למעשה מקרה מנוון **מדי** מכדי שיהיה מעניין בזכות עצמו בהקשר התיאורטי שלנו. אם ננסה להגדיר את "שפת הסודוקו" בתור שפת כל הלוחות 9×9 שממולאים באופן חלקי וניתנים להשלמה בצורה חוקית, התוצאה תהיה שפה **סופית** שכן מספרם הכולל של כל הלוחות המלאים חלקית הוא בדיוק 10^{81} (בלוח 81 משבצות ויש לנו בחירה של 10 ערכים לכל משבצת; או שתישאר ריקה, או עם מספר מ־1 עד 9). כפי שכבר ראינו, כל שפה סופית היא באופן טריוויאלי ב־P (ולכן גם ב־NP).

זוהי "בעיה" נפוצה מאוד בכל סיטואציה שבה אנו עוסקים בבעיה קונקרטית אחת שגודלה חסום. הדרך להתמודד עם הסיטואציה היא **הכללה** של הבעיה לגודל לא חסום. במקרה של סודוקו, לכל $n\in\mathbb{N}$ ניתן להגדיר בעיית סודוקו הכוללת לוח בגודל $n^2\times n^2$ כך שיש למלא אותו במספרים מהתחום $\{1,2,\ldots,n^2\}$ כך שכל שורה ועמודה כוללת את כל המספרים מקבוצה זו, ובנוסף לכך מחלקים את הלוח ל־n^2 תתי־ריבועים מגודל n^2 כל אחד שגם בהם צריכים כל המספרים להיות שונים זה מזה. סודוקו "רגיל" מתקבל עבור $n=3$.

בעיית הסודוקו המוכללת בבירור שייכת ל־NP; היחס יכלול זוגות (x,y) של לוח ממולא חלקית x ולוח מלא לגמרי y שזהה ל־x בכל המשבצות של x שאינן ריקות, וממולא על פי כללי הסודוקו.

4.2.4 הגדרה אלטרנטיבית - מכונות אי־דטרמיניסטיות

שם המחלקה P מגיע מהמילה Polynomial. מהיכן מגיע NP? שגיאה רווחת היא לומר שמקור השם הוא ב־Non-Polynomial, כלומר בעיות שאינן פתירות בזמן פולינומי, אך ממה שראינו עד כה ברור כי זה אינו נכון (בהמשך, כשנדבר על בעיות NP־שלמות, יהיה קצת יותר ברור מדוע יש כאלו שעושים טעות שכזו).

מקור ה־NP הוא בביטוי Nondeterministic Polynomial שמתייחס לאופן המקורי שבו הוגדרה המחלקה NP באמצעות **מכונות טיורינג אי דטרמיניסטיות**. השימוש במושג זה פחת מאוד בשנים שעברו מאז, שכן הוא מבלבל יותר וריאליסטי פחות מאשר הגדרה NP בתור "בעיות שקל לבדוק פתרונות עבורן", אך נציג גם אותו כדי להכיר את המושג. השורה התחתונה הרלוונטית היא ש**שתי ההגדרות הן שקולות** והשקילות היא מאוד פשוטה, עד שההבדלים בין שתי ההגדרות הם בעיקר אשליה אופטית.

הרעיון במ"ט אי־דטרמיניסטית הוא לאפשר למכונה בכל צעד שלה לבחור בין **שתי** דרכי פעולה אפשריות. משמעות הדבר היא שעל אותו קלט, למכונה יש חישובים פוטנציאליים אפשריים רבים. חלק מהחישובים הללו עשויים להסתיים בקבלה, חלקם בדחיה וחלקם עלולים לא להסתיים כלל. המוסכמה שאנו עובדים לפיה היא שמילה תהיה שייכת לשפת המכונה אם **קיים** מסלול חישוב אחד לפחות שמסתיים בקבלה של המילה.

הגדרה 11.4 מכונת טיורינג אי דטרמיניסטית מוגדרת בדומה למכונת טיורינג רגילה, פרט להגדרת פונקציית המעברים, שמוגדרת בתור

$$\delta : Q\times\Gamma\to\left(Q\times\Gamma\times\{L,R,S\}\right)^2$$

חישוב של המכונה הוא סדרת קונפיגורציות כך שמכל קונפיגורציה ניתן להגיע אל הבאה אליה על ידי בחירה של אחד משני אברי הזוג $\delta\left(q,\sigma\right)=\left(\left(p_1,\tau_1,X_1\right),\left(p_2,\tau_2,X_2\right)\right)$ ושינוי הקונפיגורציה בהתאם אליו.

עבור מ"ט M נסמן ב־$L\left(M\right)$ את אוסף המילים w כך שקיים חישוב של M על w שמסתיים ב־q_{acc}.

נאמר שמ"ט אי דטרמיניסטית M היא **פולינומית** אם קיים פולינום p כך ש**כל** מסלול חישוב שלה על w מסתיים לאחר לכל היותר $p\left(|w|\right)$ צעדים.

בצורה לא פורמלית, נאמר ש־M **מקבלת** את w אם היא מקבלת את w במסלול חישוב ספציפי כלשהו, ונאמר ש־M **לא מקבלת** את w אם כל מסלול חישוב של M על w מסתיים בדחיית w או שאינו מסתיים כלל.

נשים לב לכך ש־M **אינה** מכונה הסתברותית. במכונה הסתברותית, עבור כל מילה w קיימת הסתברות כלשהי ש־M תדחה או תקבל את w; עבור מכונה אי דטרמיניסטית הגדרנו ש־M תקבל את w אם קיים ולו מסלול אחד שמקבל את w.

אם מסכימים לוותר על הריאליזם, ניתן לדמיין את M בתור מכונה בעלת "מטבע קסם" כך שבכל צעד שלה היא מטילה אותו ובוחרת את המשך צעדיה לפי התוצאה; הקסם של המטבע מתבטא בכך שאם קיימת דרך כלשהי להגיע אל קבלת w, מטבע הקסם יבחר אותה.

דרך לא ריאליסטית נוספת להתבונן על M היא בתור מכונה שעובדת במספר "יקומים מקבילים", ומקבלת את המילה אם קיים ולו יקום אחד שבו היא קיבלה אותו.

בגישה ריאליסטית, ניתן לבצע סימולציה של ריצה של מכונה אי־דטרמיניסטית על ידי כך שבכל שלב של הסימולציה, אנו שומרים את הקונפיגורציה הנוכחית של כל אחד מה"יקומים המקבילים" שבהם המכונה נמצאת; בכל צעד מספר הקונפיגורציות שעלינו לשמור יוכפל ב־2. זוהי גישה בזבזנית מאוד, אך היא ניתנת לביצוע.

נדמיין כעת מכונת טיורינג רגילה שבנוסף לסרט הרגיל שלה יש לה גם סרט מיוחד הנקרא "סרט האי־דטרמיניזם" וכולל מחרוזת של 0־ים ו־1־ים אשר כבר קיימת באופן פלאי על הסרט בתחילת הריצה על w. בכל צעד שלה, המכונה קובעת את הצעד הבא שלה לא רק על פי $\left(q,\sigma\right)$ אלא גם על פי התו הנוכחי שהיא קוראת בסרט האי־דטרמיניזם; עבור 0 תתבצע בחירה אחת, ועבור 1 בחירה אחרת, ולאחר מכן המכונה תעבור לתו הבא בסרט האי־דטרמיניזם. מכונה כזו מזכירה מאוד את המודל האי דטרמיניסטי, פרט לכך שהיא מקבלת **מראש** את כל המידע האי דטרמיניסטי על סרט האי־דטרמיניזם שלה, בעוד המכונה האי־דטרמיניסטית מחליטה "על המקום" בכל צעד באיזו גישה לנקוט.

האם יש הבדל מהותי בין שתי גישות אלו? רק במקרים שבהם המכונה רצה עד אינסוף, אבל מקרים אלו אינם מעניינים ממילא שכן המכונה אינה מקבלת בהם את הקלט. דה פקטו שתי הגישות שקולות; אולם הגישה השניה, עם "סרט האי־דטרמיניזם" היא פשוט גישת היחסים הניתנים לזיהוי: עבור הזוג $\left(x,y\right)$ אפשר לחשוב על x בתור הקלט למכונה האי־דטרמיניסטית ועל y בתור תוכן סרט האי־דטרמיניזם.

זה מוביל אותנו למשפט הבא:

משפט 12.4 תהא L שפה. $L\in\mathrm{NP}$ אם ורק אם קיימת מ"ט אי דטרמיניסטית פולינומית M כך ש־$L=L\left(M\right)$.

הוכחה: בכיוון אחד, אם $L \in \mathrm{NP}$ אז קיים יחס R_L חסום פולינומית עם פולינום p וניתן לזיהוי פולינומי המגדיר את L.

נבנה מכונה א"ד M אשר פועלת כך על קלט x: ראשית, מייצרת בצורה אי־דטרמיניסטית מחרוזת y כך ש־$|y| \leq p(|x|)$, כך שכל מחרוזת העונה על מגבלת האורך הזה עשויה להיווצר. שנית, בודקת בזמן פולינומי האם $(x,y) \in R_L$. אם כן - מקבלת. אחרת - דוחה.

מכיוון ש־$x \in L$ אם ורק אם קיים y כך ש־$|y| \leq p(|x|)$ וגם $(x,y) \in R_L$, הרי ש־M תקבל את x אם ורק אם $x \in L$.

אם q הוא הפולינום החוסם את זמן בדיקת השייכות ל־R_L הרי שזמן ריצת המכונה שלנו חסום על ידי $q(|x|+|y|) = q(|x|+p(|x|))$ וזה זמן פולינומי כי פולינומים סגורים להרכבה.

בכיוון השני, אם קיימת מ"ט M אי־דטרמיניסטית כך ש־$L = L(M)$ אז נגדיר יחס R_L כך ש־$(x,y) \in R_L$ אם ורק אם y הוא סדרת בחירות אי־דטרמיניסטיות ש־M מקיימת בריצה שלה על x שמסתיימת בקבלת x.

בבירור $L = \{x \in \Sigma^* \mid \exists y \in \Sigma^* : (x,y) \in R_L\}$ על פי הגדרת הקבלה של מכונה אי־דטרמיניסטית (היא מקבלת את x אם ורק אם קיימת סדרת בחירות א"ד שמובילה למצב מקבל).

היחס R_L חסום פולינומית שכן M היא פולינומית ולכן זמן הריצה שלה (ומכאן - מספר הבחירות האי דטרמיניסטיות שהיא מבצעת) הוא פולינומי.

היחס R_L ניתן לזיהוי פולינומי שכן בהינתן (x,y) ניתן לבצע סימולציה של M כאשר הבחירות האי־דטרמיניסטיות של M נקראות ישירות מתוך y, ולענות כמוה. ■

הצגנו מכונות אי־דטרמיניסטיות בהקשר של חישובים יעילים, אבל היינו יכולים להציג אותן גם בחלקו הראשון של הקורס. הוכחה דומה לזו שראינו מראה כי מחלקת השפות המתקבלות על ידי מכונות אי דטרמיניסטיות ללא חסמי סיבוכיות זמן ריצה היא RE.

4.3 שאלת P=NP

ראינו כבר כי $\mathrm{P} \subseteq \mathrm{NP}$. השאלה האם $\mathrm{P} = \mathrm{NP}$ - כלומר, האם העובדה שבעיה ניתנת ל**וידוא יעיל** גוררת שהיא ניתנת ל**פתרון יעיל** היא שאלה פתוחה. ההשערה המקובלת בקרב רוב (אך לא כל) מדעני המחשב היא כי $\mathrm{P} \neq \mathrm{NP}$ אך אנחנו טרם יודעים כיצד להוכיח זאת.

כפי שנראה בהמשך, קיים מספר רב מאוד של בעיות, הנקראות "בעיות NP־שלמות", כך שפתרון יעיל **לאחת** מבעיות אלו יוכיח ש־P=NP. למרות שבעיות אלו נמצאות במרכזם של תחומים רבים, ונעשים נסיונות בלתי פוסקים לשפר את האלגוריתמים שבהם אנו משתמשים לפתרונן, עד היום לא נמצא לצורך כך אף אלגוריתם יעיל מספיק במקרה הגרוע כדי להראות שבעיה כזו שייכת ל־P. זה מטיל צל על ההיתכנות של $\mathrm{P} = \mathrm{NP}$.

מן העבר השני, מדוע קשה להוכיח ש־$\mathrm{P} \neq \mathrm{NP}$ אם זהו המצב? ראינו את הוכחתו של טיורינג לכך ש־$\mathrm{R} \neq \mathrm{RE}$; נסיונות להשתמש באותה הוכחה, או בוריאציות עליה, עבור $\mathrm{P} \neq \mathrm{NP}$ נידונות לכישלון; קיימות הוכחות לכך ששיטות כאלו (לאחר שיוגדר היטב מה הכוונה ב"שיטות כאלו") אינן מסוגלות להוכיח ש־$\mathrm{P} \neq \mathrm{NP}$. התחושה המקובלת היא שעל מנת להתמודד עם $\mathrm{P} \neq \mathrm{NP}$ אנו זקוקים לגישה חדשה במתמטיקה, כזו שעדיין אינה בהישג ידינו, בדומה לאופן שבו תעלומת "המשפט האחרון של פרמה" שנולדה במאה ה־17 נפתרה רק עם כלים מתמטיים של המאה ה־20.

כזכור, ראינו בחלקו הקודם של הקורס כי אם יחס הוא ניתן לזיהוי, אז הוא ניתן לחיפוש. ההוכחה הייתה "בזבזנית" מבחינת זמן ריצה, אך הדבר לא הפריע לנו. אותה "בזבזנות" בהקשר הנוכחי שלנו מתבטאת בכך שהשאלה האם זיהוי יעיל גורר חיפוש יעיל היא **שקולה** לשאלת $\mathrm{P} = \mathrm{NP}$. החיסכון המפתיע ב"בזבזנות" ש־$\mathrm{P} = \mathrm{NP}$ יאפשר לנו, יחול גם על בעיית החיפוש היעיל.

משפט 13.4 $\mathrm{P} = \mathrm{NP}$ אם ורק אם כל יחס S חסום פולינומית הניתן לזיהוי יעיל, ניתן לחיפוש יעיל.

הוכחה: (כיוון אחד): נניח כי כל יחס S חסום פולינומית הניתן לזיהוי יעיל, ניתן לחיפוש יעיל ונוכיח כי $\mathrm{P} = \mathrm{NP}$. תהא $L \in \mathrm{NP}$, אז קיים יחס R_L הניתן לזיהוי יעיל, חסום פולינומית ומקיים $L = \{x \in \Sigma^* \mid \exists y \in \Sigma^* : (x,y) \in R_L\}$.

על פי ההנחה שלנו, מכיוון ש־R_L ניתן לזיהוי יעיל, הוא ניתן לחיפוש יעיל. תהא M מכונה שמבצעת חיפוש יעיל שכזה. נשים לב לכך ש־$L(M) = L$ בשל האופן המדויק שבו הגדרנו את מושג החיפוש היעיל:

- אם $x \in L$ אז קיים y כך ש־$(x,y) \in R_L$. על פי הגדרתה, M תסיים את החישוב על x במקרה זה במצב q_{acc} (ובנוסף לכך תחזיר פלט, אך איננו מתעניינים בו כאן שכן אנו רק צריכים לקבל או לדחות את x).
- אם $x \notin L$ אז על פי הגדרתה, M מסיימת את החישוב על x במצב q_{rej}, כלומר דוחה את x.

מכאן ש־$L(M) = L$, ובנוסף לכך M פולינומית, כך שקיבלנו ש־$L \in \mathrm{P}$. ■

כיוון זה של ההוכחה היה פשוט יחסית, אבל הכיוון השני מורכב יותר ולכן לפני שנוכיח אותו נפתח עם אינטואיציה בסיוע משחק הסודוקו שכבר ראינו.

כשחושבים על סודוקו כבעיית חיפוש, הקלט x הוא לוח מלא בחלקו במספרים, והפלט y הוא אותו לוח, כאשר כל המשבצות הריקות מולאו בצורה חוקית. השפה SUDOKU שמוגדרת על ידי בעיית הסודוקו היא שפת כל הלוחות החלקיים ש**ניתן להשלים** לפתרון מלא.

בבירור $\text{SUDOKU} \in \text{NP}$ בסיוע היחס שתיארנו זה עתה. לכן, אם $\text{P} = \text{NP}$, נזכה ביכולת לקבוע, על ידי מבט על לוח חלקי, אם ניתן להשלים אותו לפתרון מלא. כיצד יכולת זו עוזרת לנו **למצוא** פתרון?

בהינתן הלוח x, נפעל כך: ראשית נבדוק האם $x \in \text{SUDOKU}$, כלומר האם x פתיר בכלל, אחרת אפשר פשוט לדחות. כעת, משידוע לנו ש־x פתיר, נעבור סדרתית על כל המשבצות הריקות ב־x. לכל משבצת כזו, ננסה להציב בה את כל המספרים מ־1 עד 9. לכל הצבה כזו נקבל לוח חדש x', וכעת נבדוק האם $x' \in \text{SUDOKU}$. כלומר, האם גם אחרי המספר שהצבנו במשבצת הלוח נותר פתיר. אם הוא נותר פתיר, נעבור הלאה אל המשבצת הבאה; אחרת נמחק את המספר שהצבנו בתא וננסה מספר אחר. מכיוון ש־x פתיר **מובטח לנו** כי לפחות אחת מההצבות תצליח ותותיר את הלוח פתיר; כעת נמשיך לפעול באותו אופן על הלוח x' שהתקבל מההצבה המוצלחת, וכן הלאה ־ עד אשר נקבל לוח מלא.

הסיבה שהאלגוריתם כולו הוא פולינומי, היא שבכל שלב של האלגוריתם אנחנו צריכים לבחור בין מספר קטן יחסית של אפשרויות (במקרה שלנו מספר קבוע: 9 אפשרויות), ואחרי כל בחירה של אפשרות אנחנו מקבלים בזמן פולינומי פידבק האם זו הייתה בחירה מוצלחת או לא. בצורה זו אנו נבדלים מהסיטואציה הרגילה, שבה אחרי בחירה כזו ייתכן שהיה עלינו להמשיך לשחק עוד ועוד, ולבצע עוד בחירות רבות, לפני שהיינו משתכנעים שהבחירה שלנו הייתה לא מוצלחת. במקרה הרגיל היה עלינו **לטייל בעומק עץ המשחק** ואילו בסיוע $\text{P} = \text{NP}$ הצלחנו "לקטום" את עץ המשחק ומספיק לנו בכל שלב להציץ צעד אחד קדימה.

סודוקו הוא דוגמא למקרה פשוט במיוחד, הסיטואציה הכללית יותר מסובכת מעט יותר, ואת זאת ניתן להדגים בעזרת בעיית המסלול ההמילטוני. הקלט לבעיית המסלול ההמילטוני הוא גרף G ותו לא; הגרף אינו כבר "פתור חלקית" כפי שקורה בסודוקו. עם זאת, ניתן להסתכל על בעיה דומה מאוד: במקום שהקלט יהיה G, הוא יהיה זוג (G, p') שכולל גרף G **והתחלה** של מסלול p' בו, כאשר השאלה שיש לענות עליה היא האם ניתן **להמשיך** את המסלול p' עד אשר יתקבל מסלול המילטוני. בעיה זו נפתרת באותה גישת חיפוש כמו בעיית הסודוקו: בכל שלב נבחר צומת אחד להוסיף למסלול הקיים, ונבדוק האם התוצאה ניתנת להשלמה למסלול המילטוני. אם כן, נמשיך עם הצומת שבחרנו ואחרת ננסה צומת אחר.

ההוכחה הפורמלית היא שילוב של שתי הטכניקות שראינו: המעבר אל בעיית הכרעה שכוללת בתוכה גם פתרון חלקי, וההרחבה הסדרתית של פתרון חלקי בעזרת חיפוש של "הצעד הבא".

הוכחה: (כיוון שני): נניח כי $\text{P} = \text{NP}$. יהא S יחס חסום פולינומית הניתן לזיהוי פולינומי; נוכיח כי הוא ניתן לחיפוש פולינומי.

נגדיר את השפה הבאה: $L = \{(x, y') \mid \exists y'' : (x, y'y'') \in S\}$.

נראה כי $L \in \text{NP}$: מכונת טיורינג אי דטרמיניסטית שבהינתן (x, y') מנחשת מחרוזת y'' שהיא פולינומית ב־$|x|$ ומפעילה את המכונה שמזהה את S על $(x, y'y'')$ בבירור מקבלת את השפה L והיא פולינומית מכיון שניחוש y'' שהוא פולינומי ב־$|x|$ הוא פולינומי ב־$|y'| + |x|$, והרצת המכונה שמזהה את S על $(x, y'y'')$ היא פולינומית ב־$|x|$.

מכיוון שהנחנו כי $\text{P} = \text{NP}$, נקבל ש־$L \in \text{P}$. תהא M_L מכונה פולינומית המכריעה את L. נראה כיצד להשתמש בה כדי לבנות מכונה M שתבצע חיפוש פולינומי של היחס S.

בהינתן קלט x, המכונה M תבצע את התהליך האיטרטיבי הבא:

1. (בדיקה ראשונית): הרץ את M_L על (x, ε). אם M_L דחתה, דוחים (עוברים למצב q_{rej}).
2. (אתחול) $y \leftarrow \varepsilon$.
3. (תנאי עצירה) בדוק האם $(x, y) \in S$ באמצעות המכונה המזהה את S. אם כן, הוצא את הפלט y ועבור למצב q_{acc}.
4. (צעד): לכל $\sigma \in \Sigma$:
   (א) הרץ את M_L על $(x, y\sigma)$. אם M_L קיבלה, קבע $y \leftarrow y\sigma$ וחזור לשלב 3.

לא הגדרנו מה המכונה תעשה בשלב 4 אם M_L דחתה את **כל** ה־y'־ים האפשריים, כי כפי שנראה בקרוב המצב הזה אינו יכול להתקיים.

נוכיח כי המכונה M פותרת את בעיית החיפוש של S.

ראשית, אם לא קיים y כך ש־$(x, y) \in S$ אז מהגדרת L נקבל ש־$(x, \varepsilon) \notin L$ ולכן M תעבור ל־q_{rej}, כנדרש.

הדרך הנוספת שבה M עשויה לעצור היא בשלב 3. מכיוון שזה מתבצע רק אם $(x, y) \in S$ כאשר y הוא פלט המכונה בשלב זה, ברור ש**אם** המכונה עוצרת, היא עונה נכון. מה שנותר להראות הוא שאם קיים z כך ש־$(x, z) \in S$ אז המכונה תעצור אחרי מספר פולינומי של צעדים.

נראה כי האלגוריתם מקיים את האינוריאנטה הבאה: בתחילת שלב 3, תמיד מתקיים $(x, y) \in L$.

ראשית, בפעם הראשונה שבה הגענו לשלב 3 זה היה אחרי שבשלב 1 בדקנו ישירות ש־$(x,\varepsilon)\in L$ ובשלב 2 קבענו $y=\varepsilon$ כך שהטענה מתקיימת בפעם הראשונה הזו.

כעת נראה ששלב 4 באמת מצליח: דהיינו שבשלב 4 מוצאים σ כך ש־$(x,y\sigma)\in L$ ואז מכיוון שקובעים $y\leftarrow y\sigma$ בבירור יתקיים $(x,y)\in L$ גם בתחילת האיטרציה הבאה.

אם בשלב 3 אנחנו מגלים ש־$(x,y)\in S$ אז אנחנו עוצרים ומקבלים ולא מגיעים אל שלב 4, כלומר ניתן להניח ש־$(x,y)\notin S$ אבל מכך ש־$(x,y)\in L$ נובע שקיימת סיפא y' כך ש־$(x,yy')\in S$. מאחר ו־$(x,y)\notin S$ אז $y'\neq\varepsilon$ כלומר ניתן לכתוב $y'=\sigma y''$. מכיוון ש־$(x,y\sigma y'')\in S$ נובע ש־$(x,y\sigma)\in L$ ולכן בשלב 4 הבדיקה תצליח עבור σ (היא עשויה להצליח עוד קודם עבור אות אחרת).

ראינו שבכל איטרציה של האלגוריתם מתקיים אחד משני דברים: או שהאלגוריתם עוצר ב־q_{acc} עם פלט תקין, או שהוא מגדיל את y בתו אחד. מכיוון ש־S חסום פולינומית, נגיע תוך מספר פולינומי של צעדים ל־y מאורך מקסימלי ובשלב זה בהכרח נקבל (כי לא ייתכן שנגדיל את y עוד יותר). זה מסיים את ההוכחה. ■

כעת נוכל להציג טענה דומה ופשוטה יותר, שעוסקת בחישוב פונקציות:

טענה 14.4 תהא $f:\Sigma^*\to\Sigma^*$ פונקציה מלאה. אז $f\in\mathrm{POLY}$ אם ורק אם f חסומה פולינומית (כלומר $|f(x)|\le p(|x|)$ עבור פולינום p כלשהו) וגם $L'_f\in\mathrm{P}$ כאשר $L'_f=\{(x,y')\ \mid\exists y''\in\Sigma^*:f(x)=y'y''\}$.

הוכחה: הרעיון דומה מאוד להוכחה שכבר ראינו. מצד אחד, אם $f\in\mathrm{POLY}$ ברור שהיא חסומה פולינומית כי מ"ט עם חסם זמן ריצה p שרצה על קלט x יכולה להוציא לפלט לכל היותר $p(|x|)$ תווים. בנוסף, כדי לבדוק האם $(x,y')\in L'_f$ בזמן פולינומי פשוט מחשבים את $f(x)$ ובודקים האם y' סיפא של התוצאה.

הכיוון השני הוא המסובך יותר: אם f חסומה פולינומית ו־$L'_f\in\mathrm{P}$, אז כדי לחשב את $f(x)$ נפעל כך: נגדיר $y'=\varepsilon$ ובאופן אינדוקטיבי, נבדוק לכל $\sigma\in\Sigma$ האם $y'\sigma\in L'_f$. אם מצאנו σ כזה, נגדיר $y'\leftarrow y'\sigma$ ונמשיך; אם לא מצאנו σ כזה אז $f(x)=y'$. בסך הכל נדרשות $|f(x)|$ איטרציות שבכל אחת מהן בודקים לכל היותר $|\Sigma|$ אפשרויות כשכל בדיקה מתבצעת בזמן פולינומי ־ בסך הכל פולינומי. ■

5 בעיות NP־שלמות

5.1 מבוא

בחלקו הראשון של הקורס ראינו שתי מחלקות: R,RE ובחלקו השני ראינו שתי מחלקות: P,NP. היחס בין אברי הזוג הראשון זהה ליחס בין אברי הזוג השני, אך כפי שראינו ־ יש לנו הוכחה פשוטה ש־$\mathrm{R}\neq\mathrm{RE}$ אך אין לנו הוכחה ש־$\mathrm{P}\neq\mathrm{NP}$.

בחלקו הראשון של הקורס ראינו כי $\mathrm{HP}\notin\mathrm{R}$ ואז הוכחנו באמצעותה עבור שפות נוספת שהן אינן ב־R על ידי כך שהראינו **רדוקציה** $\mathrm{HP}\le L$. בחלק השני של הקורס לא ברור מאליו מה הגרסה המקבילה ל־HP שבה נרצה להשתמש ־ אם אין לנו שפה ב־NP שאנו יודעים שאינה ב־P, מה כן יש לנו? מתברר שעדיין יש לנו משהו ־ שפות כך **שאם** $\mathrm{P}\neq\mathrm{NP}$ אז **בודאות** הן אינן שייכות ל־P.

נפתח בהגדרת התכונה המדויקת של שפות אלו שתעניין אותנו ולאחר מכן נציג דוגמאות רבות. גם כאן, כמו בחלק הראשון של הקורס, הכלי המרכזי שבו נשתמש יהיה **רדוקציה**, אלא שעלינו להתאים את ההגדרה להקשר של זמן הריצה הפולינומי שמאפיין את P.

הגדרה 1.5 יהיו $L_1,L_2\subseteq\Sigma^*$ שפות כלשהן. **רדוקציה פולינומית** מ־L_1 אל L_2 היא פונקציה $f:\Sigma^*\to\Sigma^*$ כך ש־$f\in\mathrm{POLY}$ המקיימת

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

אם קיימת רדוקציה פולינומית מ־L_1 אל L_2 מסמנים זאת $L_1\le_p L_2$.

כמו בחלקו הראשון של הקורס, כך גם כאן ־ רדוקציות פולינומיות הן מעניינות בזכות משפט הרדוקציה המתאים:

משפט 2.5 תהיינה L_1,L_2 שפות כך ש־$L_1\le_p L_2$

- אם $L_2\in\mathrm{P}$ אז $L_1\in\mathrm{P}$.

- אם $L_2 \in \mathrm{NP}$ אז $L_1 \in \mathrm{NP}$.

הוכחה: תהא M_f המכונה שמחשבת את הרדוקציה.

אם $L_2 \in \mathrm{P}$ עם מ"ט M_2 פולינומית, אז מ"ט פולינומית M_1 עבור L_1, על קלט x תחשב את $f(x)$ ותריץ את M_2 על $f(x)$.

אם p הוא הפולינום שחוסם את זמן ריצת M_f ו־q הוא הפולינום שחוסם את זמן ריצה M_2, אז $|f(x)| \le p(|x|)$ וזהו גם חסם על זמן חישוב $f(x)$. זמן הרצת M_2 על $f(x)$ חסום על ידי $q(|f(x)|) = q(p(|x|))$ שהוא פולינומי ב־$|x|$ כי הרכבת הפולינומים p, q היא פולינום. מכאן ש־M_1 היא פולינומית.

אם $L_2 \in \mathrm{NP}$ אז קיים יחס S_2 חסום פולינומית עם הפולינום q וניתן לזיהוי פולינומי כך ש־$L_2 = \{x \mid \exists y : (x,y) \in S_2\}$. נגדיר יחס S_1 על ידי $S_1 = \{(x,y) \mid (f(x), y) \in S_2\}$.

היחס חסום פולינומית כי $(f(x), y) \in S_2$ גורר ש־$|y| \le q(|f(x)|)$ וכפי שראינו $|f(x)| \le p(|x|)$ כך ש־$|y| \le q(p(|x|))$.

היחס ניתן לזיהוי פולינומי כי בהינתן (x,y) נחשב בזמן פולינומי את $f(x)$ ואז נבדוק בזמן פולינומי אם $(f(x), y) \in S_2$.

היחס מגדיר את L_1 כי בשל תקפות הרדוקציה f אנו יודעים ש־$x \in L_1$ אם ורק אם $f(x) \in L_2$ אם ורק אם $(f(x), y) \in S_2$ אם ורק אם $(x,y) \in S_1$. ■

ראינו כבר שרדוקציות מקיימות טרנזיטיביות. מכיוון שתכונה זו תהיה שימושית מאוד עבורנו בהמשך, נראה זאת במפורש גם במקרה הנוכחי:

משפט 3.5 אם $L_1 \le_p L_2$ וגם $L_2 \le_p L_3$ אז $L_1 \le_p L_3$.

הוכחה: יהיו f, g הרדוקציות של $L_1 \le_p L_2$ ו־$L_2 \le_p L_3$, בהתאמה. נתבונן על הפונקציה המורכבת gf. היא ניתנת לחישוב בזמן פולינומי על ידי חישוב f ואז הפעלת המכונה לחישוב g על התוצאה; הזמן הפולינומי נובע מכך שהרכבת פולינומים היא פולינום. תקפות הרדוקציה נובעת מכך ש־$g(f(x)) \in L_3 \iff f(x) \in L_2 \iff x \in L_1$. ■

נעבור כעת להגדרה המרכזית שלנו.

הגדרה 4.5 שפה L נקראת NP־שלמה ונסמן זאת $L \in \mathrm{NPC}$ אם

- $L \in \mathrm{NP}$
- לכל $L' \in \mathrm{NP}$ מתקיים $L' \le_p L$

על כל שפה שמקיימת את התנאי השני אנו אומרים שהיא **NP־קשה**. המושג "NP־שלמה" בא לתאר שפות שהן גם NP־קשות **וגם** שייכות בעצמן ל־NP.

במילים אחרות, שפה היא NP־שלמה אם היא "בדרגת הקושי הגבוהה ביותר של שפות ב־NP" במובן זה שפתרון יעיל שלה גורר קיום פתרון יעיל לכל בעיה ב־NP:

משפט 5.5 אם L היא שפה NP־שלמה אז $L \in \mathrm{P}$ אם ורק אם $\mathrm{P} = \mathrm{NP}$.

הוכחה: אם $\mathrm{P} = \mathrm{NP}$ אז מכיוון ש־$L \in \mathrm{NP}$ נקבל מייד ש־$L \in \mathrm{P}$. בכיוון השני, אם $L \in \mathrm{P}$, תהא $L' \in \mathrm{NP}$ כלשהי. מהגדרת שפה NP־שלמה נקבל ש־$L' \le_p L$ וממשפט הרדוקציה נקבל ש־$L' \in \mathrm{P}$. ■

כיצד ניתן להראות ששפה L היא NP־שלמה? ההגדרה נראית מאתגרת למדי - יש להראות כי **כל** שפה ב־NP, מורכבת ומתוחכמת ככל שתהיה, ניתנת לרדוקציה אל L. ואכן, ההוכחה שנראה בהמשך תהיה מורכבת למדי.

עם זאת, יש לנו דרך קיצור משמעותית: אם כבר ידוע לנו על שפה NP־שלמה, ניתן להיעזר בה כדי להוכיח עבור שפות אחרות שהן גם כן NP־שלמות:

משפט 6.5 אם L_1 היא שפה NP־שלמה ו־$L_2 \in \mathrm{NP}$ ומתקיים $L_1 \le_p L_2$ אז גם L_2 היא NP־שלמה.

הוכחה: מכיוון ש־$L_2 \in \mathrm{NP}$ נותר רק להראות שכל שפה ב־NP ניתנת לרדוקציה אליה. תהא $L \in \mathrm{NP}$ כלשהי. מכיוון ש־L_1 היא NP־שלמה, אז $L \le_p L_1$. מכיוון ש־$L_1 \le_p L_2$ נקבל מטרנזיטיביות ש־$L \le_p L_2$. ■

זוהי סיטואציה מקבילה לזו שהייתה בחלקו הראשון של הקורס. שם הוכחנו שאם $L_1 \notin \mathrm{R}$ וגם $L_1 \le L_2$ אז גם $L_2 \notin \mathrm{R}$; גם כאן הרדוקציה היא מהשפה ש"ידוע שהיא קשה" אל השפה ש"עדיין לא ידוע עליה כלום".

5.2 דוגמאות ראשונות לשפות NP־שלמות

5.2.1 השפות SAT ו־k-SAT

השפה SAT מעניינת אותנו גם בגלל השימושיות שלה באופן כללי, וגם בגלל העובדה שהיא מהווה את "נקודת המוצא" שלנו להוכחה ששפות הן NP־שלמות, בזכות המשפט שמראה שהיא NP־שלמה ללא צורך ברדוקציה משפה אחרת:

משפט 7.5 (משפט קוק לוין) $\mathrm{SAT}\in\mathrm{NPC}$.

מכיוון שהוכחת המשפט מורכבת, נמתין איתה להמשך. לבינתיים נציג את הגדרת השפה. SAT היא שפת כל הנוסחאות בתחשיב הפסוקים שנמצאות בצורת CNF וספיקות; איננו מניחים ידע מוקדם עם מושגים אלו ולכן נציג את הידע הרלוונטי במפורט.

הגדרה 8.5 (נוסחאות CNF והגדרת הספיקות עבורן)

- **משתנה** הוא איבר של קבוצה כלשהי. לרוב נסמן משתנים באותיות כמו x,y,z וכדומה.
- **ליטרל** l הוא או משתנה x או הביטוי $\neg x$ כאשר אנו חושבים על הסימן $\neg$ כמייצג "שלילה".
- **פסוקית CNF** C היא ביטוי מהצורה $(l_1\vee l_2\vee\ldots\vee l_k)$ כך שכל l_i הוא ליטרל. אנו חושבים על הסימן $\vee$ כמייצג "או".
- **פסוק CNF** φ הוא ביטוי מהצורה $C_1\wedge C_2\wedge\ldots\wedge C_n$ כך שכל C_i הוא פסוקית CNF. אנו חושבים על הסימן $\wedge$ כמייצג "וגם".
- **השמה** α היא פונקציה שמתאימה לכל משתנה ערך מהקבוצה $\{\mathrm{T},\mathrm{F}\}$ (אנו חושבים על אבריה בתור "אמת" ו"שקר").
- השמה **מספקת** ליטרל l אם הליטרל הוא x וההשמה נתנה למשתנה x את הערך T או שהליטרל הוא $\neg x$ וההשמה נתנה למשתנה x את הערך F.
- השמה **מספקת** פסוקית CNF $C=(l_1\vee l_2\vee\ldots\vee l_k)$ אם היא מספקת **לפחות אחד** מהליטרלים $l_1,l_2,\ldots,l_k$.
- השמה **מספקת** פסוק CNF $\varphi=C_1\wedge C_2\wedge\ldots\wedge C_n$ אם היא מספקת את **כל** הפסוקיות $C_1,C_2,\ldots,C_n$.
- פסוק CNF φ הוא **ספיק** אם קיימת השמה α שמספקת אותו.

כעת ניתן להגדיר את השפה SAT:

הגדרה 9.5 SAT היא שפת כל פסוקי ה־CNF הספיקים.

לדוגמא, הפסוק $(x_1\vee\neg x_3)\wedge(x_3\vee\neg x_5\vee x_2)\wedge(x_5)$ ספיק על ידי ההשמה הבאה:

$$\begin{aligned}\alpha(x_1)&=\mathrm{T}\\ \alpha(x_2)&=\mathrm{T}\\ \alpha(x_3)&=\mathrm{F}\\ \alpha(x_5)&=\mathrm{T}\end{aligned}$$

קל לראות כי $\mathrm{SAT}\in\mathrm{NP}$: היחס שכולל זוגות (φ,α) של פסוק והשמה מספקת עבורו מוכיח את השייכות ל־NP. השמה ניתנת לתיאור בעזרת סדרה של n ביטים כאשר n הוא מספר המשתנים השונים המופיעים ב־φ ולכן כמובן שהיא פולינומית ב־$|\varphi|$ (היא יותר קצרה מ־$|\varphi|$) ובדיקת שהשמה מספקת פסוק היא פולינומית (עוברים על כל פסוקית ולכל ליטרל בה בודקים אם הוא מקבל את הערך המתאים בהשמה).

עם זאת, אלגוריתם נאיבי לבדיקה האם פסוק ספיק שעובר על כל ההשמות יכלול במקרה הגרוע 2^n בדיקות (ניתן להראות שקיימים פסוקים ספיקים עם השמה מספקת יחידה). קיימים אלגוריתמים מורכבים יותר לבדיקת ספיקות; למעשה, קיים תחום שלם של אלגוריתמים הנקראים SAT Solvers שמשתמשים בהיוריסטיקות חכמות לבדיקת ספיקות. אלגוריתמים אלו

הופכים פתרון בעיות SAT לפרקטי במקרים רבים, אבל הם אינם מציעים שיפור אסימפטוטי לזמן הריצה הגרוע ביותר ולכן אינם רלוונטיים למה שנעשה כאן.

שפה שימושית נוספת עבורו היא השפה 3SAT שכוללת את כל פסוקי ה־CNF הספיקים שבהם בכל פסוקית יש **בדיוק** שלושה ליטרלים. שפה זו פשוטה יותר מ־SAT ולכן יהיה יותר קל לבצע רדוקציות ממנה לשפות אחרות; עם זאת, היא מורכבת דיו כדי להיות NP־שלמה בעצמה:

משפט 10.5 $\mathrm{SAT}\le_p 3\mathrm{SAT}$

הוכחה: יהא φ פסוק CNF. נראה כיצד לבנות פסוק φ' שבו כל פסוקית היא בדיוק בעלת שלושה ליטרלים, וכך ש־φ ספיק אם ורק אם φ' ספיק.

נראה כיצד מטפלים בכל פסוקית של φ לחוד. תהא $C=(l_1\vee\ldots\vee l_n)$ פסוקית כלשהי של φ.

אם ב־C רק ליטרל אחד או שניים, פשוט נשכפל את הליטרלים הללו שוב עד לקבלת שלושה ליטרלים בפסוקית. כלומר, אם $C=(l_1)$ אז ל־φ' נוסיף את הפסוקית $(l_1\vee l_1\vee l_1)$ ואם $C=(l_1\vee l_2)$ אז נוסיף את הפסוקית $(l_1\vee l_2\vee l_2)$. בבירור השמה מספקת את C אם ורק אם היא מספקת את ההרחבה שלה.

אם ב־C יש בדיוק שלושה ליטרלים, אפשר להוסיף אותה כמות שהיא אל φ'.

נותר לטפל במקרה שבו מספר הליטרלים n ב־C גדול מ־3. במקרה זה, יהיו $y_1,y_2,\ldots,y_{n-3}$ משתנים חדשים שלא הופיעו ב־φ ולא השתמשנו בהם עד כה.

כעת נחליף את הפסוקית $(l_1\vee\ldots\vee l_n)$ ב"שרשרת" הפסוקיות הבאה:

$$(l_1\vee l_2\vee y_1)\wedge(\neg y_1\vee l_3\vee y_2)\wedge(\neg y_2\vee l_4\vee y_3)\wedge\ldots\wedge(\neg y_{n-3}\vee l_{n-1}\vee l_n)$$

זו רדוקציה פולינומית שכן כל פסוקית הוחלפה לכל היותר בשרשרת של $O(n)$ פסוקיות ־ גודל התוצאה עדיין פולינומי.

נראה כעת כי φ ספיק אם ורק אם φ' ספיק.

בכיוון אחד, נניח ש־φ ספיק באמצעות השמה α ונמצא השמה α' שתספק את φ'. α' תהיה זהה ל־α על המשתנים של φ; נראה כיצד להגדיר אותה על המשתנים החדשים שהוספנו.

תהא $C=(l_1\vee\ldots\vee l_n)$ פסוקית ב־φ עם $n>3$ משתנים. מכיוון ש־C מסתפקת תחת α קיים בה ליטרל שמסתפק; נסמנו l_i. כעת נגדיר את α' על המשתנים $y_1,\ldots,y_{n-3}$ שהוספנו כשהמרנו את הפסוקית הזו אל שרשרת הפסוקיות ב־φ' באופן הבא:

$$\alpha'(y_k)=\begin{cases}\mathrm{T} & k\le i-2\\ \mathrm{F} & k>i-2\end{cases}$$

ההשמה מספקת את כל הפסוקיות בשרשרת: הפסוקיות שהופיעו בהן $y_1,y_2,\ldots,y_{i-2}$ הסתפקו כי משתנים אלו קיבלו T והפסוקיות שהופיעו בהן $\neg y_{i-1},\ldots,\neg y_{n-3}$ הסתפקו כי המשתנים של ליטרלים אלו קיבלו F. נותרה רק פסוקית אחד שאינה כוללת לא את אלו ולא את אלו: הפסוקית $(\neg y_{i-2}\vee l_i\vee y_{i-1})$, שמסתפקת בזכות העובדה ש־α מספקת את l_i, או הפסוקית $(l_1\vee l_2\vee y_1)$ (אם $i=1,2$) שמסתפקת בצורה דומה, או $(\neg y_{n-3}\vee l_{n-1}\vee l_n)$ (אם $i=n-1,n$) שגם היא מסתפקת בצורה דומה.

פסוקיות עם 3 או פחות ליטרלים מסתפקות על ידי הערכים ש־α נתנה למשתנים המקוריים. בנוסף, לא יכולה להיות התנגשות באופן שבו אנחנו מגדירים את α' בהתאם לפסוקיות שונות של φ שכן לכל פסוקית הוספנו קבוצה חדשה של משתנים. מכאן קיבלנו ש־α' מספקת את φ'.

בכיוון ההפוך, אם α' היא השמה שמספקת את φ' אז נגדיר α הזהה ל־α' על המשתנים של φ. בבירור α מספקת כל פסוקית מגודל עד 3 ב־φ'. תהא $C=(l_1\vee\ldots\vee l_n)$ פסוקית עם $n>3$ של φ; נראה כיצד α בהכרח מספקת גם אותה. נתבונן על שרשרת הפסוקיות שמתאימה לפסוקית זו ב־φ':

$$(l_1\vee l_2\vee y_1)\wedge(\neg y_1\vee l_3\vee y_2)\wedge(\neg y_2\vee l_4\vee y_3)\wedge\ldots\wedge(\neg y_{n-3}\vee l_{n-1}\vee l_n)$$

אם $\alpha(y_1)=\mathrm{F}$ אז כדי שתסתפק $(l_1\vee l_2\vee y_1)$ בהכרח α מספקת את l_1 או l_2 וסיימנו. נניח אם כן ש־$\alpha(y_1)=\mathrm{T}$. אם כל יתר המשתנים מקבלים T גם כן ובפרט y_{n-3} אז כדי שתסתפק $(\neg y_{n-3}\vee l_{n-1}\vee l_n)$, בהכרח α מספקת את l_{n-1} או l_n וסיימנו. אם לא כל המשתנים מקבלים T יהא y_i המשתנה עם האינדקס הקטן ביותר שקיבל F. זה אינו y_1 שכבר הנחנו שמקבל T, ולכן גם y_{i-1} מקבל T כי האינדקס שלו קטן משל y_i. כעת נתבונן בפסוקית $(\neg y_{i-1}\vee l_{i+1}\vee y_i)$, אז בפסוקית זו הליטרל הראשון והאחרון מקבלים שניהם F ומכאן שבהכרח l_{i+1} מסתפק, וסיימנו. ■

את מה שעשינו עבור 3 ניתן לעשות באופן כללי - להגדיר את השפה kSAT בתור שפת כל פסוקי ה־CNF הספיקים שבהם כל פסוקית כוללת בדיוק k ליטרלים. אותה הוכחה שראינו עבור 3SAT תעבוד לכל שפה אחרת עם $k > 3$, אבל עבור 2SAT היא בבירור לא תעבוד - ה"שרשרת" התבססה על עיטוף כל ליטרל מהפסוקית המקורית בשני משתנים חדשים. כישלון הרדוקציה אינו מקרי, שכן אפשר להראות ש־2SAT שייכת ל־P:

משפט 11.5 $2SAT \in P$.

הוכחה: האינטואיציה מאחורי ההוכחה היא שניתן לחשוב על פסוקית כמייצגת **גרירה**. הפסוקית $(x \vee y)$ שקולה לוגית לפסוקים הלוגיים $\neg x \to y$ ו־$\neg y \to x$. האינטואיציה הזו משמשת כדי לבנות, בהינתן פסוק 2CNF, "גרף גרירות" עבורו: גרף שבו יש צומת לכל משתנה x ושלילתו $\neg x$ ויש חץ $l_1 \to l_2$ אם בפסוק יש את הפסוקית $(\neg l_1 \vee l_2)$ (כאשר אנו מתייחסים אל $\neg\neg$ כמבטל את עצמו).

כעת ניתן להוכיח כי פסוק ה־2CNF הוא ספיק אם ורק אם לא קיים משתנה x כך שבגרף הגרירות קיים מסלול מ־x אל $\neg x$ **וגם** מסלול מ־$\neg x$ אל x. בדיקת מסלולים כאלו בגרף שייכת ל־P. ■

5.2.2 השפה Vertex Cover

בעיית **הכיסוי בצמתים** (Vertex Cover) היא בעיה בתורת הגרפים:

הגדרה 12.5 יהא $G = (V, E)$ גרף פשוט ולא מכוון. קבוצה $B \subseteq V$ היא **כיסוי בצמתים** של G אם לכל $(u, v) \in E$ מתקיים ש־$u \in B$ או $v \in B$. במילים אחרות: כל קשת בגרף פוגשת את הקבוצה B.
השפה VC תוגדר באופן הבא:

$$\mathrm{VC} = \{(G, k) \mid \exists B \subseteq V : B \text{ is vertex cover} \wedge |B| \le k\}$$

בבירור $\mathrm{VC} \in \mathrm{NP}$; היחס הוא של זוגות $((G, k), B)$ כך ש־B הוא כיסוי בצמתים מהגודל הנכון.

כזכור, גרף הוא **פשוט** אם אין בו קשתות מקבילות (יותר מקשת אחת בין זוג צמתים) וחוגים עצמיים (קשתות מצומת לעצמה). ההגבלה שלנו ש־G יהיה פשוט אינה מהותית; בהינתן גרף לא פשוט שאנו רוצים למצוא עבורו כיסוי בצמתים, ברור שעלינו להוסיף לכיסוי כל צומת בעלת חוג עצמי, ואפשר למחוק את כל הקשתות המקבילות בין זוג צמתים למעט אחת. לאחר "תיקונים" אלו נישאר עם גרף פשוט. הסיבה שבגללה אנו מסתפקים בגרפים פשוטים היא כדי לפשט רדוקציות מ־VC לשפות אחרות.

משפט 13.5 $3SAT \le_p VC$.

נציג את ההוכחה בצורה פורמלית, אך לפני כן נצביע על הרעיון. אנחנו רוצים לקודד פסוק CNF בתור גרף, כך שתהיה התאמה בצורה כלשהי בין קבוצת צמתים שנבחרת מהגרף ובין השמה עבור הפסוק. לשם כך, הגרף שלנו יכלול שתי קבוצות של צמתים: קבוצת צמתים שמקודדת **ליטרלים** וקבוצת צמתים שמקודדת **מופעים בפסוקיות של ליטרלים**. בחירה של צומת מקבוצת הליטרלים פירושה יהיה שזה הליטרל שאנו בוחרים לספק, ובחירה של צומת מקבוצת המופעים של ליטרלים בפסוקיות פירושה יהיה שזה הליטרל שאנו משתמשים בו כדי לספק את הפסוקית שבה הוא מופיע (מבין שלושת הליטרלים האפשריים).

כדי להשיג את האפקט המבוקש הזה, נהנדס בקפידה את הגודל המקסימלי k של הכיסוי בצמתים שאנו מאפשרים, כך שלא ניתן יהיה להוסיף לכיסוי כמה איברים שנרצה, אלא נהיה חייבים **לכל** משתנה לבחור אך ורק משני הליטרלים שמתאימים לו, **ולכל** פסוקית לבחור אך ורק אחד משלושת הליטרלים המופיעים בה (למעשה, כפי שנראה מייד, "בחירה" כזו תתבטא בכך שלכיסוי יתווספו שני הליטרלים האחרים שבפסוקית).

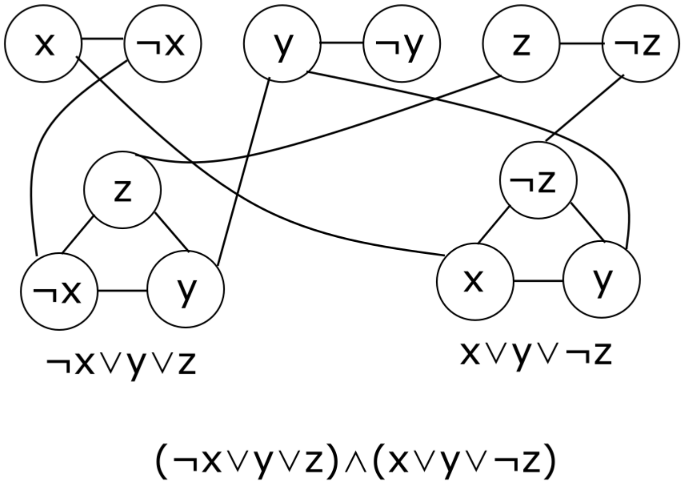
הוכחה: נציג רדוקציה המעבירה פסוק 3CNF φ לזוג (G_φ, k) באופן הבא:

תהא $\{x_1, x_2, \ldots, x_n\}$ קבוצת המשתנים שמופיעים ב־φ ו־$C_1, C_2, \ldots, C_m$ קבוצת הפסוקיות שלו. נגדיר $k = n + 2m$ ואת הגרף G_φ נבנה באופן הבא:

לכל משתנה x שמופיע ב־φ, נוסיף לגרף את הצומת x ואת הצומת $\neg x$ ונחבר את שניהם בקשת. נכנה צמתים אלו **צמתי המשתנים**.

לכל פסוקית $C = (l_1 \vee l_2 \vee l_3)$ נוסיף לגרף שלושה צמתים שיסומנו ב־l_1, l_2, l_3 ויחוברו כולם זה לזה, כך שנוצרת צורת משולש. בנוסף, נחבר כל ליטרל אל צומת המשתנה המתאים לו. כלומר, אם הליטרל הוא מהצורה x נחבר אותו אל הצומת x בצמתי המשתנים; אם הוא מהצורה $\neg x$ נחבר אותו אל הצומת $\neg x$ בצמתי המשתנים.

נציג איור לדוגמא של הבניה:



הרדוקציה בבירור פולינומית כי מספר הצמתים שאנו מוסיפים לכל משתנה ופסוקית ב־φ הוא קבוע וההוספה שלהם אינה דורשת חישוב.

נעבור להוכיח את תקפות הרדוקציה. ראשית, נניח ש־φ ספיק על ידי השמה α ונציג כיסוי בצמתים מגודל $n+2m$ של G_φ. לכל משתנה x, אם $\alpha(x)=\mathrm{T}$ אז נוסיף לכיסוי את צומת המשתנה x, ואחרת נוסיף לכיסוי את צומת המשתנה $\neg x$. לכל פסוקית $C=(l_1\vee l_2\vee l_3)$ מכיוון ש־α מספקת את φ קיים ליטרל ב־C שהיא מספקת, נסמנו ב־l_i. אז מבין שלושת צמתי הפסוקית שהוספנו לגרף, נוסיף לכיסוי שלנו את שני הצמתים **שאינם** l_i.

בבירור גודל הכיסוי שלנו הוא $n+2m$ כי הוספנו בדיוק צומת אחד לכל אחד מ־n המשתנים, ושני צמתים לכל אחת מ־m הפסוקיות.

תהא כעת קשת כלשהי בגרף. יש שלוש אפשרויות:

1. הקשת מחברת שני צמתי משתנים, ולכן מחברת צומת x לצומת $\neg x$ עבור משתנה x מסויים. על פי בניית הכיסוי שלנו, לכל משתנה x אחד מהצמתים $x,\neg x$ התווסף לכיסוי ולכן קשת זו מכוסה.
2. הקשת היא חלק מה"משולש" שמחבר את צמתי הליטרלים בפסוקית כלשהי. מכיוון שלקחנו לכיסוי שני צמתים מבין אברי המשולש, הקשת בהכרח מכוסה (קיים רק צומת אחד במשולש שלא נלקח לכיסוי, אבל הקשת מחוברת לשני צמתים מהמשולש).
3. הקשת מחברת בין צומת משתנה ובין צומת של ליטרל בפסוקית. במקרה זה ישנן שתי אפשרויות: אם הוספנו את צומת הליטרל לכיסוי, סיימנו; אחרת, בהכרח הצומת הזה מתאים לליטרל שמסתפק תחת α ולכן צומת המשתנה שאליו הוא מחובר כן שייך לכיסוי, וגם במקרה זה סיימנו.

נוכיח כעת את הכיוון השני ־ כלומר, שאם קיים כיסוי בצמתים של G_φ מגודל לכל היותר $n+2m$ אז φ ספיק. נניח כי קיים כיסוי כזה בצמתים, אז:

1. לכל זוג $x,\neg x$ של צמתי משתנים, אחד משניהם חייב להשתייך לכיסוי עקב הקשת שמחברת אותם. בסך הכל לפחות n צמתים חייבים להתווסף כך לכיסוי.
2. לכל פסוקית $C=(l_1\vee l_2\vee l_3)$ לפחות שניים מצמתי הליטרלים של הפסוקית חייבים להשתייך לכיסוי כי משולש לא ניתן לכסות עם צומת בודד (הקשת בין שני הצמתים שלא נבחרו לא תהיה מכוסה). בסך הכל לפחות $2m$ צמתים חייבים להתווסף כך לכיסוי.

אם בשלב 1 או בשלב 2 יילקחו **יותר** מאשר ה־n וה־$2m$ צמתים האפשריים, גודל הכיסוי יהיה **גדול יותר** מאשר $n+2m$, בסתירה לחסם $k=n+2m$ שהוא חלק מפלט הרדוקציה. מכאן נסיק שבשלב 1 לוקחים **בדיוק** צומת אחד מכל זוג, ובשלב 2 לוקחים **בדיוק** זוג צמתים מכל שלשה.

נגדיר את ההשמה α בצורה הבאה: לכל משתנה x, אם צומת המשתנה x שייך לכיסוי אז $\alpha(x)=\mathrm{T}$ ואחרת $\alpha(x)=\mathrm{F}$. נוכיח כי זוהי השמה מספקת:

תהא $C=(l_1\vee l_2\vee l_3)$ פסוקית כלשהי. כפי שראינו, בדיוק שני צמתים המתאימים לפסוקית התווספו לכיסוי. נתבונן על הצומת שלא התווסף לכיסוי; צומת זה מחובר לצומת משתנה המתאים לאותו ליטרל l_i כמו זה של הצומת, וחייב להשתייך לכיסוי אחרת הקשת הזו לא תהיה מכוסה. על פי ההגדרה שלנו את α, הליטרל l_i מסתפק תחת α, ולכן C הסתפקה, כמבוקש. ■

5.2.3 השפה Hitting Set ("קבוצה מייצגת")

עם השפה Hitting Set אנו עוברים מעולמות הלוגיקה והגרפים לעולם של קבוצות. תהא X קבוצה סופית כלשהי, ויהיו $A_1,A_2,\ldots,A_n\subseteq X$ תתי־קבוצות של X. אז Hitting Set עבור אוסף תתי הקבוצות הוא תת־קבוצה $B\subseteq X$ כך ש־$A_i\cap B\neq\emptyset$ לכל $1\le i\le n$.

למשל, נניח שבידינו אוסף שירים (X) וכמה ז'אנרים של מוזיקה, כך ששיר מהאוסף יכול להשתייך למספר ז'אנרים (הקבוצות $A_1,A_2,\ldots,A_n$). מטרתנו היא להכין רשימת השמעה (B) שבה כל ז'אנר מופיע לפחות פעם אחת (אותו שיר יכול לייצג ז'אנרים שונים בבת אחת).

כמובן, תמיד ניתן לקחת את כל אברי X ולקבל Hitting Set ולכן האתגר הוא למצוא קבוצה B שתהיה קטנה יחסית.

הגדרה 14.5 השפה Hitting Set מוגדרת על ידי

$$\mathrm{HS}=\left\{(A_1,\ldots,A_n,k)\ \mid\ \exists B\subseteq\bigcup_{i=1}^{n}A_i:|B|\le k\wedge B\text{ is a hitting set}\right\}$$

בבירור $\mathrm{HS}\in\mathrm{NP}$: היחס פשוט נותן את הקבוצה B, שהיא פולינומית בגודל $|\bigcup_{i=1}^{n}A_i|$ שכן היא תת־קבוצה של איחוד זה.

טענה 15.5 $\mathrm{VC}\le_p\mathrm{HS}$.

רעיון ההוכחה פשוט למדי. בעצם, VC היא מקרה פרטי של HS כאשר כל הקבוצות הן מגודל 2. מכאן הרדוקציה כמעט מיידית:

הוכחה: בהינתן קלט (G,k) עבור VC כך ש־$G=(V,E)$ עם $E=\{e_1,e_2,\ldots,e_n\}$, הרדוקציה תעביר את (G,k) אל $((e_1,e_2,\ldots,e_n,k))$. זוהי בבירור רדוקציה פולינומית.

על מנת לראות את תקפות הרדוקציה נשים לב לכך שתת־קבוצה $E'\subseteq E$ היא כיסוי בצמתים של G אם ורק אם E' היא Hitting Set של $(e_1,\ldots,e_n)$. ■

5.2.4 השפה Set Cover ("כיסוי בקבוצות")

השפה Set Cover היא מעין מקרה משלים ל־Hitting Set. אם במקרה הקודם רצינו תת־קבוצה של X ש"תופסת את כל הקבוצות", עכשיו אנחנו רוצים תת קבוצה של אוסף הקבוצות ש"תופסת את כל X". נגדיר זאת פורמלית.

הגדרה 16.5 תהא X קבוצה ו־$A_1,\ldots,A_n\subseteq X$ תת־קבוצות שלה. אז Set Cover של X היא אוסף $A_{i_1},\ldots,A_{i_k}$ כך ש־$\bigcup_{t=1}^{k}A_{i_t}=X$.

השפה SC מוגדרת כך:

$$\mathrm{SC}=\left\{(A_1,\ldots,A_n,k)\ \mid\ \exists i_1,\ldots,i_k:\bigcup_{t=1}^{k}A_{i_t}=\bigcup_{j=1}^{n}A_j\right\}$$

בבירור $\mathrm{SC}\in\mathrm{NP}$ כי היחס פשוט כולל את הקבוצות שב־Set Cover.

משפט 17.5 $\mathrm{VC}\le_p\mathrm{SC}$.

בדומה ל־HS גם פה ההוכחה פשוטה כי VC הוא מקרה פרטי פשוט של SC, אך מעט יותר מאתגר לראות זאת. הרעיון הוא שבמקרה זה, $X=E$, ואילו A_i הוא אוסף כל הקשתות שמכוסות על ידי הצומת v_i.

נניח שבגרף $G=(V,E)$ עם n צמתים, הצמתים מסומנים בתור $V=\{v_1,v_2,\ldots,v_n\}$.
הרדוקציה תוגדר בתור

$$(G,k)\mapsto(A_1,A_2,\ldots,A_n,k)$$

כך שלכל $v_i\in V$ אנו מגדירים $A_i=\{e\in E\mid v_i\in e\}$.

הרדוקציה פולינומית כי יצירת כל A_i דורשת מעבר יחיד על קבוצת הקשתות, כך שיש לנו $O\left(|V||E|\right)$ מעברים בסך הכל לצורך חישוב הרדוקציה.

נראה את תקפות הרדוקציה. נשים לב לכך שב־G יש כיסוי בצמתים $U\subseteq V$ אם ורק אם הקבוצה $\{i_t\mid v_{i_t}\in U\}$ היא Set Cover: בכיוון אחד, אם U הוא כיסוי בצמתים, תהא $e\in E$ אז קיים $v_i\in U$ שמכסה אותה, ולכן $e\in A_i$ ועל פי ההגדרה, A_i שייכת ל־Set Cover. בכיוון השני, אם $\{i_t\mid v_{i_t}\in U\}$ היא Set Cover ו־$e\in E$ אז לפי הגדרה קיים t כך ש־i_t שייך ל־Set Cover ו־$e\in A_{i_t}$, אבל אז מהגדרת הקבוצה, $A_{i_t}\in U$ ולכן e מכוסה על ידי U.

5.2.5 השפה $\text{IP}01$ (תכנון לינארי 01 בשלמים)

בעיות **תכנון לינארי** הן סוג נפוץ ושימושי מאוד של בעיות אופטימיזציה. בבעיה כזו נתונה **פונקציית מטרה** שמקבלת וקטור של ערכים ומחזירה מספר ממשי, והמטרה שלנו היא למצוא את הקלט שעבורו הפונקציה מחזירה פלט מקסימלי, בהינתן **אילוצים** על הקלטים האפשריים. המילה **לינארי** מגיעה מכך שהן פונקציית המטרה והן האילוצים הם לינאריים, כלומר מערבים רק צירופים לינאריים של המשתנים.

בעיית תכנון לינארי על n משתנים נתונה על ידי מטריצה $A\in\mathbb{R}^{m\times n}$ המתארת m אילוצים ווקטור $c\in\mathbb{R}^{1\times n}$ כך שהמטרה היא למקסם את הפונקציה $f\left(x\right)=c\cdot x$ בהינתן האילוץ $Ax\ge\overline{0}$ (כאן $\ge$ בהשוואת שני וקטורים פירושו שמשווים כניסה־כניסה).

קיימת תורה עשירה של שיטות לפתרון בעיות תכנון לינארי, ובעיות אלו ניתנות לפתרון יעיל; אולם **הגבלה** על הקלטים שיכולים להתקבל כך שנדרש מהם להיות מספרים שלמים הופכת את הבעיה ל־NP־קשה (זה הד לתופעה כללית במתמטיקה לפיה בעיות מעל שדה כמו $\mathbb{R}$ הן קלות יותר לפתרון מאשר מעל חוג כמו $\mathbb{Z}$).

אנו נתמקד במקרה פרטי ־ כזה שבו $A\in\mathbb{Z}^{m\times n}$, הערכים המותרים לקלטים הם 0 ו־1 בלבד, ואנו לא מעוניינים למקסם את פונקציית המטרה אלא רק לבדוק האם תחת האילוצים היא יכולה להחזיר ערך מעל לסף מסויים. כדי לפשט את הסימונים, ניתן לאחד את האילוצים על הקלטים עם האילוץ על פונקציית המטרה, ולקבל את הפורמליזם הבא של הבעיה:

הגדרה 18.5 (תכנות 01 בשלמים) בהינתן מטריצה $A\in\mathbb{Z}^{m\times n}$ ווקטור $b\in\mathbb{Z}^m$, יש לקבוע האם קיים $x\in\{0,1\}^n$ כך ש־$Ax\ge b$. השפה 01IP כוללת את כל ה־(A,b) שמקיימים זאת.

כרגיל, קל לראות ש־$01\text{IP}\in\text{NP}$ פשוט על ידי יחס שבו ה־x נתון.

משפט 19.5 $\text{VC}\le_p 01\text{IP}$.

הוכחה: אנחנו רוצים לקודד בעיית VC בצורה כלשהי כך ש**פתרון** של בעיית ה־01IP ימדל פתרון של בעיית ה־VC. מכיוון שפתרון של בעיית 01IP הוא פשוט סדרה של 0 ו־1, אינטואיטיבי לחשוב שהפתרון הזה יתאר את הצמתים של G שמתווספים לכיסוי (מקבלים 1) אל מול אלו שאינם מתווספים לכיסוי (מקבלים 0). האם אפשר לתאר את האילוצים של בעיית VC בעזרת משוואות לינאריות?

ראשית, $x_1+x_2+\ldots+x_n$ שווה בדיוק למספר הצמתים שנלקחו לכיסוי. אנו רוצים לקודד את האילוץ

$$x_1+x_2+\ldots+x_n\le k$$

לרוע המזל, הגדרת 01IP תומכת באי־שוויונים דווקא בכיוון ההפוך, של $\ge$, אבל קל לפתור זאת על ידי כפל שני האגפים ב־-1 לקבלת האילוץ השקול

$$-x_1-x_2-\ldots-x_n\ge -k$$

הדרישה הנוספת שלנו היא שלכל קשת $e = (v_i, v_j)$ **לפחות אחד** מבין שני הצמתים המחוברים לקשת יהיה שייך לכיסוי, כלומר המשתנה שלו יקבל 1. זה מתורגם אל האילוץ

$$x_i + x_j \geq 1$$

או, בכתיב מלא:

$$0 \cdot x_1 + 0 \cdot x_2 + \cdots + x_i + \ldots + x_j + \ldots + 0 \cdot x_n \geq 1$$

אם כן, נקבל $A \in \mathbb{Z}^{|E|+1\times n}$ כך שהשורה הראשונה ב־A כולה -1 ושאר שורות A כוללות זוג 1־ים במקומות i, j לכל $(v_i, v_j) \in E$, ו־$b \in \mathbb{Z}^{|E|+1}$ כך ש־$b = (-k, 1, \ldots, 1)$.
הבניה של A, b דורשת מעבר יחידה על E ולכן היא פולינומית, והוכחת נכונות הרדוקציה נובעת מההגדרה. ■

5.3 הוכחות ישירות לכך ששפות הן NP־שלמות

5.3.1 השפה Bounded Halting

פתחנו עם הטענה ש־SAT היא שפה NP־שלמה ודחינו את ההוכחה להמשך עקב הקושי שלה, אך למעשה קל מאוד לתאר במפורש שפה NP־שלמה ולהוכיח שהיא כזו; השפה Bounded Halting שנציג כעת. לרוע המזל, זו אינה שפה שקל לבצע ממנה רדוקציות, ומכאן החשיבות של SAT.

השפה Bounded Halting היא למעשה וריאציה על בעיית העצירה עם שני הבדלים: ראשית, המכונה שמתקבלת כקלט היא **אי דטרמיניסטית**. שנית, בנוסף למכונה וקלט מצורף גם חסם זמן מפורש על מספר הצעדים של המכונה. חסם הזמן מיוצג בייצוג אונרי ולא בינארי כדי שגודל הקלט יהיה פרופורציוני לחסם זמן הריצה (אם הוא היה מיוצג בבינארי, חסם זמן הריצה היה אקספוננציאלי בגודלו ביחס לגודל הקלט):

הגדרה 20.5 השפה Bounded Halting היא

$$\mathrm{BH} = \left\{ \left(\langle M \rangle, x, 1^t \right) \mid M \text{ has a halting path on } x \text{ in } t \text{ steps} \right\}$$

משפט 21.5 $\mathrm{BH} \in \mathrm{NPC}$

הוכחה: ראשית נראה כי BH שייכת אל NP. זאת באמצעות יחס $\left(\left(\langle M \rangle, x, 1^t\right), y\right)$ כך ש־y היא סדרת בחירות הא"ד ש־M מבצעת על x עד שהיא עוצרת תוך לכל היותר t צעדים. היחס חסום פולינומית כי $|y| \leq t = |1^t|$ והוא ניתן לבדיקה פולינומית על ידי מכונה שמריצה את M על x בהתאם לבחירות שמתוארות ב־y.

כעת נראה כי BH היא NP־שלמה. תהא $L \in \mathrm{NP}$ כלשהי. נראה רדוקציה $L \leq_p \mathrm{BH}$.

ראשית, מכיוון ש־$L \in \mathrm{NP}$ אנו יודעים שקיימים שני אובייקטים:

- מ"ט פולינומית אי דטרמיניסטית M כך ש־$L(M) = L$.
- פולינום $p(x)$ שהוא חסם זמן הריצה של M.

פונקציית הרדוקציה f_L תפעל כך: $f_L(x) = \left(\langle M' \rangle, x, 1^{p(|x|)}\right)$, כך ש־$M'$ היא מכונה זהה ל־M למעט העובדה שבמקום כניסה ל־q_{rej} המכונה נכנסת למצב של לולאה אינסופית. את $1^{p(|x|)}$ ניתן לחשב בזמן פולינומי מתוך x, זאת מכיוון שאורך $p(|x|)$ הוא פולינומי ב־x, והפולינום p עצמו, כמו גם M', שניהם חלק מקידוד המכונה שמחשבת את הרדוקציה f_L.

מכיוון ש־$x \in L$ אם ורק אם קיים ל־M מסלול מקבל עליו תוך $p(|x|)$ צעדים אם ורק אם קיים ל־M' מסלול שעוצר עליו תוך $p(|x|)$ צעדים, סיימנו. ■

5.3.2 משפט קוק־לוין

עכשיו, לאחר שראינו מספר רדוקציות ואת העקרונות הכלליים שמאחוריהם, ננסה את כוחנו בהוכחת משפט קוק־לוין:

משפט 22.5 (משפט קוק־לוין) השפה SAT היא NP־שלמה.

שייכות SAT ל־NP היא ברורה; האתגר הוא להראות כי היא NP־שלמה שלא דרך רדוקציה משפה אחרת. כלומר, בהינתן $L\in\mathrm{NP}$ עלינו להראות רדוקציה $L\le\mathrm{SAT}$, ובמילים אחרות - לכל $w\in\Sigma^*$ עלינו להחזיר פסוק φ_w כך ש־$w\in L$ אם ורק אם φ_w הוא פסוק ספיק.

מכיוון ש־$L\in\mathrm{NP}$, קיים יחס R_L כך ש־$w\in L$ אם ורק אם קיים y כך ש־$(w,y)\in R_L$. הבדיקה האם $(w,y)\in R_L$ מתבצעת בידי מכונת טיורינג פולינומית M, ואת הריצה של מכונה כזו על (w,y) אפשר לתאר בתור סדרת קונפיגורציות. מכיוון שיש לנו חסם על זמן הריצה של המכונה, יש לנו חסם על הגודל המקסימלי האפשרי של קונפיגורציה עבורה, כך שמלכתחילה אפשר לחשוב על כל הקונפיגורציות כאילו הן מאותו אורך. כעת אפשר לדמיין סידור של כל הקונפיגורציות בטבלה: בכל שורה נמצאת קונפיגורציה, כך שכל עמודה מתארת תו אחד מהקונפיגורציה (תו כזה הוא או אות σ שנמצאת על הסרט, או זוג (q,σ) שמתאר גם את מצב המכונה ומלמד אותנו שהראש הקורא מצביע על המיקום הזה).

הרעיון במשפט קוק הוא שהשמות למשתנים של φ_w ניתנות יהיו לתרגום לטבלה כזו של קונפיגורציות. אנו נבנה את φ_w בצורה כזו שמבטיחה שההשמה תהיה מספקת רק אם:

1. ההשמות אכן מתארות טבלה חוקית.
2. השורה הראשונה בטבלה מתאימה לקונפיגורציה ההתחלתית של M על (w,y) עבור y כלשהו שהוא פולינומי ב־w.
3. קיימת בטבלה שורה שבה המכונה נמצאת במצב q_{acc}.
4. כל זוג שורות סמוכות בטבלה מתארות קונפיגורציות עוקבות, כלומר המעבר משורה אחת לבאה מתבצע בהתאם להגדרות של M.

האופן שבו φ_w תיבנה יבטיח שלרוב המשתנים ב־φ_w "אין ברירה" בשאלה מה הערך שלהם יהיה; הוא ייקבע באופן מוחלט בידי ערכם של משתנים אחרים, ואם לא יתאים לקביעה הזו, φ_w לא יסתפק. עם זאת, המשתנים ב־φ_w שמקודדים את y יישארו חופשיים; זה יוביל לכך שקיים y כך ש־$(w,y)\in R_L$ אם ורק אם קיימת השמה שמספקת את φ_w.

הסיבה שבגללה הבניה הזו יכולה לעבוד נעוצה באספקט טכני אחד של מכונות טיורינג, שהוא מה שנותן לנו את ההוכחה כולה: השינוי שמתבצע בין זוג קונפיגורציות סמוכות הוא **לוקלי**. רוב התאים מועתקים כמות שהם, והמקומות היחידים שבהם משהו עשוי להשתנות הם סביב הראש הקורא של המכונה. הפשטות הזו תאפשר ל־φ_w להיות פשוט יחסית, ולכן פולינומי בגודלו.

נעבור להוכחה הפורמלית:

הוכחה: מכיוון ש־$L\in\mathrm{NP}$ קיים יחס R_L שהוא חסום פולינומית על ידי הפולינום p, ניתן לזיהוי פולינומי על ידי מכונת טיורינג שזמן הריצה שלה חסום בידי פולינום q, ומקיים $L=\{w\in\Sigma^*\mid\exists y\in\Sigma^*:(w,y)\in R_L\}$.

בהינתן $w\in\Sigma^*$ נבנה פסוק CNF φ_w באופן הבא:

ראשית, נניח בלי הגבלת הכלליות שהשפה L מקודדת כך שהזוג (w,y) מיוצג פורמלית על ידי $w\#y$ כך ש־$\#\in\Sigma$ הוא סימן שאינו שייך לאף w,y המקיימים $(w,y)\in R_L$ (כל שפה ב־NP ניתן להעביר לצורה כזו על ידי הוספת סימן חדש לא"ב ושימוש בו בתור תו מפריד).

בפרט, $|(w,y)|=|w|+1+|y|$.

כעת, נשים לב לכך שלכל y עבורו ש־$(w,y)\in R_L$ מהגדרת חסם פולינומי, $|y|\le p(|w|)$. זמן ריצת M חסום על ידי הפולינום q ולכן על הקלט (w,y) זמן ריצה זה חסום על ידי $q(|(w,y)|)=q(n+1+p(n))$ כך ש־$n=|w|$, וערך זה הוא פולינומי ב־n.

נסמן $N=q(n+1+p(n))+1$. כעת אנו יודעים על N שני דברים:

1. זהו **מספר הקונפיגורציות המקסימלי** בכל ריצה של M על (w,y) כנ"ל (מספר הקונפיגורציות הוא מספר הצעדים + 1 ולכן הוספנו 1 בהגדרת N).
2. זהו **האורך המקסימלי של קונפיגורציה** בכל ריצה של M על (x,y) כנ"ל, כי התא הימני ביותר שהראש יכול להגיע אליו במספר הצעדים המקסימלי הוא N.

מכאן שאם קיים חישוב של M על זוג (w,y) שמסתיים בקבלה, אפשר לתאר אותו באמצעות טבלה $N\times N$ שמתארת את סדרת הקונפיגורציות שמתאימה לחישוב.

נסמן $\Delta\triangleq\Gamma\cup(Q\times\Gamma)$: אלו הסימנים שיכולים להופיע כחלק מקונפיגורציה (או אות, או זוג של מצב ואות).

לכל $i,j\in\{1,2,\ldots,N\}$ נגדיר קבוצת משתנים שמתארת את תוכן התא ה־j בקונפיגורציה ה־i, באופן הבא:

לכל $a\in\Delta$ נוסיף משתנה בוליאני $X_{i,j}^{a}$. אנו חושבים על הצבת T במשתנה זה בתור הטענה "בתא ה־j בקונפיגורציה ה־i מופיע a" והצבת F בתור הטענה שדבר זה אינו קורה. אלו יהיו המשתנים היחידים שבהם נשתמש בפסוק שלנו.

לכל i,j נוסיף כעת אל φ_w את הפסוקיות הבאות, שמבטיחות שבתא i,j יופיע לפחות דבר אחד, ולא יופיעו בו שני דברים שונים:

$$\left(\bigvee_{a\in\Delta}X_{i,j}^{a}\right)$$

$$\forall a\neq b\in\Delta:\left(\neg X_{i,j}^{a}\vee\neg X_{i,j}^{b}\right)$$

בסך הכל לכל i,j אנו מוסיפים פסוקית אחת מאורך $|\Delta|$ ו־$O\left(|\Delta|^{2}\right)$ פסוקיות מאורך 2. מכיוון ש־$|\Delta|$ קבוע ולא תלוי באורך $|w|$ (הוא נובע ישירות מהמכונה M) האורך הכולל של מה שהוספנו הוא $O(1)$. מכיוון שאנו מוסיפים זאת לכל i,j נקבל בסך הכל אורך כולל של $O\left(N^{2}\right)$, שעדיין פולינומי ב־$|w|$ שכן N פולינומי ב־$|w|$.

כעת קיימת התאמה חח"ע ועל בין השמות של φ_w שמספקות את הפסוקיות שהוספנו עד כה, ובין טבלאות מסדר $N\times N$ מעל אברי Δ: שלב 1 בתיאור שהצגנו קודם.

נעבור לפסוקיות שמבטיחות שהקונפיגורציה הראשונה (השורה $i=1$) תהיה קונפיגורציה התחלתית חוקית של M על הזוג (w,y). כזכור, הנחנו בלי הגבלת הכלליות כי (w,y) מיוצג על ידי $w\#y$ כך ש־$\#$ לא מופיע ב־w,y.

נסמן $w=w_1w_2\cdots w_n$ כך ש־$w_k\in\Delta$ לכל $1\le k\le n$. כעת נוסיף ל־φ_w את סדרת הפסוקיות בנות איבר בודד הבאות:

$$X_{1,1}^{(q_0,w_1)}\wedge X_{1,2}^{w_2}\wedge X_{1,3}^{w_3}\wedge\ldots\wedge X_{1,n}^{w_n}$$

פסוקיות אלו מבטיחות שתחילת הקונפיגורציה תהיה מהצורה $(q_0,w_1)\,w_2w_3\cdots w_n$, כלומר שהקונפיגורציה מתחילה ב־w.

נוסיף כעת ל־φ_w פסוקית בת משתנה בודד:

$$X_{1,n+1}^{\#}$$

פסוקית זו מבטיחה שאחרי w יבוא $\#$.

כעת נעבור לייצוג של y. הערכים של y לא נקבעים באופן יחיד - למעשה, אלו יהיו המשתנים היחידים שהם "חופשיים" במובן זה שניתן להציב בהם ערכים שונים מבלי שערכם ינבע ישירות ממשתנים אחרים. עם זאת, עדיין יש שתי מגבלות על y:

1. $|y|\le p(|w|)$. כלומר, לא ניתן לנצל את **כל** התאים שנותרו עד תא N בשביל y, אלא רק את $m=p(|w|)$ התאים הראשונים שאחרי תא $n+1$.
2. $y\in\Sigma^{*}$, כלומר לא כל תו של Δ יכול להופיע ב־y. למעשה, מכיוון שאנו מניחים ש־$\#$ לא מופיע ב־y, הדרישה שלנו היא חזקה יותר: $y\in(\Sigma\backslash\{\#\})^{*}$.

אם כן, לכל $n+1<j\le n+1+m$ ולכל $\sigma\notin(\Sigma\backslash\{\#\})$ נוסיף פסוקית בת משתנה בודד

$$\neg X_{1,j}^{\sigma}$$

לבסוף, כל יתר הסרט מעבר לתא $n+1+m$ הוא $\flat$, אז לכל $n+1+m<j$ נוסיף פסוקית

$$X_{1,j}^{\flat}$$

השלמנו את שלב 2 בתיאור שהצגנו קודם. הוספנו פסוקיות מאורך כולל של פחות מ־$N\cdot|\Delta|$, כך ש־φ_w נותר פולינומי בגודלו.

נעבור אל שלב 3. על מנת להבטיח שקיימת בטבלה שורה שבה מופיע q_{acc} נוכל פשוט לבדוק את כל תאי הטבלה. נוודא שלא מופיע q_{rej} בשום שלב, וש־q_{acc} מופיע לפחות פעם אחת. בצורה זו מובטח שהחישוב שמתואר על ידי הטבלה אכן יסתיים במצב מקבל (ולא, נאמר, יסתיים קודם במצב q_{rej} ו־q_{acc} יופיע בהמשך באופן חסר משמעות).

נוסיף ל־φ_w את הפסוקיות הבאות:

$$\left(\bigvee_{1\le i,j\le N,\sigma\in\Gamma}X_{i,j}^{(q_{acc},\sigma)}\right)$$

$$\bigwedge_{1\le i,j\le N,\sigma\in\Gamma}\neg X_{i,j}^{(q_{rej},\sigma)}$$

הוספנו פסוקיות מאורך כולל של $O\left(N^2\cdot|\Gamma|\right)=O\left(N^2\right)$ - עדיין פולינומי. בכך סיימנו את שלב 3.

נותר להוסיף פסוקיות שמבטיחות שהמעבר בין שתי קונפיגורציות עוקבות הוא תקין.

בהינתן זוג קונפיגורציות עוקבות C,C' שמסודרות כך שהשורה של C' נמצאת מעל השורה של C, כל תא של C' נקבע באחת מהדרכים הבאות:

- אם התא שמתחתיו כלל (q,σ) ובמכונה M $\delta\left(q,\sigma\right)=\left(p,\tau,S\right)$. במקרה זה תוכן התא יהיה (p,τ).
- אם התא שמתחתיו כלל (q,σ) ובמכונה M $\delta\left(q,\sigma\right)=\left(p,\tau,X\right)$ כך ש־$X\in\{L,R\}$. במקרה זה תוכן התא יהיה τ.
- אם התא שמתחתיו כלל σ והתא שמימין לתא זה כלל (q,σ') ובמכונה M $\delta\left(q,\sigma'\right)=\left(p,\tau,L\right)$. במקרה זה תוכן התא יהיה (p,σ).
- אם התא שמתחתיו כלל σ והתא שמשמאל לתא זה כלל (q,σ') ובמכונה M $\delta\left(q,\sigma'\right)=\left(p,\tau,R\right)$. במקרה זה תוכן התא יהיה (p,σ).
- אם התא שמתחתיו כלל σ והתאים משמאל ומימין לתא זה לא כוללים את הראש הקורא של המכונה. במקרה זה תוכן התא יהיה σ.

במילים אחרות, תוכן כל תא הוא פונקציה של שלושת התאים הסמוכים אליו בשורה מתחת: זה שמתחתיו ואלו שמימין ומשמאל לזה שמתחתיו. לכל תא יש $|\Delta|$ משתנים שמתארים אותו, כך שבסך הכל עבור כל תא, הפסוק שמתאר את תקינות התא בהקשר של פונקציית המעברים כולל $|\Delta|^4$ משתנים. אופי הפסוק הזה תלוי מאוד בפונקציית המעברים δ של המכונה, אבל ניתן לכתוב **כל פסוק** בעזרת נוסחת CNF. אם הפסוק הוא על n משתנים, גודל נוסחת ה־CNF הזו עשוי להיות 2^n, כך שבמקרה זה גודל הפסוק עלול להיות $2^{|\Delta|^4}$. על פניו זה מספר גדול, אולם זוהי הנקודה המרכזית בהוכחה: $|\Delta|$ הוא **קבוע** שאינו תלוי בגודל w אלא רק בתכונות המכונה M. לכן $2^{|\Delta|^4}$ הוא קבוע.

אם כן, לכל $1\le i,j\le N$ יש לנו פסוק מגודל קבוע שמתאר את תקינות התא (i,j) ביחס לפונקציית המעברים δ, ולכן בסך הכל גודל הפסוק שאנו מוסיפים אל φ_w הוא $O\left(N^2\right)$ - פולינומי ב־$|w|$.

בכך הסתיימה בניית φ_w, מה שסיים את הוכחת משפט קוק־לוין; מהבניה שלנו עולה ש־$w\in L$ אם ורק אם φ_w ספיק, כנדרש. ■

5.4 דוגמאות מתקדמות לשפות NP־שלמות

5.4.1 בעיית סכום תת־הקבוצה (Subset Sum)

בבעיית Subset Sum הקלט כולל סדרה $a_1, a_2, \ldots, a_n$ של מספרים טבעיים ומספר יעד k, והשאלה היא האם קיימת תת־סדרה של הסדרה המקורית שמסתכמת אל k. פורמלית:

$$\mathrm{SS} = \left\{ (a_1, \ldots, a_n, k) \ \mid \exists I \subseteq [n] : \sum_{i \in I} a_i = k \right\}$$

כרגיל, בבירור $\mathrm{SS} \in \mathrm{NP}$ עם היחס שכולל את I המתאימה.

משפט 23.5 $\mathrm{VC} \le_p \mathrm{SS}$

ההוכחה אינה קשה אבל הצורך לעבוד עם מספרים טבעיים מוסיף קושי טכני, אז נפתח עם תיאור אינטואיטיבי של המתרחש בה. יהא $G = (V, E)$ גרף ונסמן $|V| = n$ ו־$|E| = m$. בהינתן k עלינו לבדוק האם קיים כיסוי בצמתים מגודל קטן או שווה מ־k; מכיוון שתמיד ניתן להוסיף לכיסוי עוד צמתים ועדיין לקבל כיסוי, מספיק לבדוק אם קיים כיסוי שגודלו **בדיוק** k.

נקודד כל צומת $v_i \in V$ בעזרת וקטור $a_i \in \{0,1\}^m$ כך ש־ $[a_i]_j = \begin{cases} 1 & v_i \in e_j \\ 0 & v_i \notin e_j \end{cases}$. כעת, אם $I \subseteq [n]$ היא קבוצה של צמתים, אז $\sum_{i \in I} a_i$ יהיה וקטור שבו כל כניסה (המתאימה לקשת e_j כלשהי) תהיה או 2 (אם שני הצמתים של e_j ב־I), או 1 (אם רק אחד מהם שם) או 0 (אם אף אחד מהם אינו שם). במילים אחרות, I הוא כיסוי בצמתים אם ורק אם וקטור הסכום לא כולל כניסות שהן 0.

אנו רוצים לפשט מעט את העניינים כך שאם I הוא כיסוי בצמתים, אז אפשר לקבל וקטור תוצאה שבו כל הכניסות הן **בדיוק** 2. לצורך כך נוסיף עוד וקטורי עזר, שכל אחד מהם יכול להוסיף 1 לכניסה בודדת: וקטורים מהצורה $b_j \in \{0,1\}^m$ עבור $j \in [m]$ כך ש־$[b_j]_t = \delta_{jt}$. כעת, בהינתן I שמהווה סיכוי בצמתים, אפשר על ידי הוספת b־ים מתאימים להגיע אל 2 בכל הכניסות (לכניסות שכבר יש בהן 2 לא נזדקק ל־b, ולכניסות שיש בהן 1 נזדקק לו). מכיוון ש־b יכול להוסיף רק 1 לכניסה, הרי שכניסה שהיה בה קודם 0 לא תוכל להפוך ל־2.

סיכום ביניים: ב־G קיים כיסוי בצמתים אם ורק אם קיימת תת־קבוצה של ה־$a_1, \ldots, a_n, b_1, \ldots, b_m$ שסכומה הוא הוקטור $\{2\}^m$.

מה עדיין חסר? האילוץ של k איברים בדיוק בכיסוי. לצורך כך נוסיף עוד כניסה אחת אחרונה לוקטורים שלנו, שתהיה 1 בכל וקטורי ה־a ו־0 בכל וקטורי ה־b, כך שערך הכניסה הזו בסכום סופר כמה וקטורי a השתתפו בו.

כדי לעבור מהניסוח ה"וקטורי" לניסוח עם מספרים טבעיים המתאים ל־SS עלינו רק לשים לב שאפשר לחשוב על מספר טבעי בתור וקטור של ספרות, בזכות ההתאמה $(d_1, d_2, \ldots, d_t) \mapsto \sum_{i=1}^{t} d_i 10^{i-1}$. **הוכחה:** יהא (G, k) כלשהו כך ש־$G = (V, E)$ עם $|V| = n, |E| = m$. נבנה קלט לבעיית SS: $(a_1, \ldots, a_n, b_1, \ldots, b_m, t)$ כך ש:

- $a_i = 10^m + \sum_{j: v_i \in e_j} 10^{j-1}$
- $b_j = 10^{j-1}$
- $t = k \cdot 10^m + \sum_{j=1}^{m} 2 \cdot 10^{j-1}$

הבניה בבירור פולינומית כי חישוב הסכומים הנ"ל מתבצע בזמן פולינומי.

נעבור להוכחת נכונות.

ראשית, נניח כי קיים ב־G כיסוי בצמתים מגודל לכל היותר k, ולכן בפרט קיים אחד מגודל k בדיוק. כלומר קיימת $I \subseteq [n]$ כך ש־$|I| = k$ ולכל $e \in E$ קיים $i \in I$ כך ש־$v_i \in e$.

נגדיר $J = \{j \in [m] \mid |\{v_i \mid i \in I \land v_i \in e_j\}| = 1\}$.

כעת, הפתרון לבעיית ה־SS כולל את כל ה־a_i־ים עם אינדקסים ב־I וה־b_j־ים עם אינדקסים ב־J. כדי לראות זאת נתבונן על הסכום

$$\begin{aligned}\sum_{i\in I} a_i + \sum_{j\in J} b_j &= \sum_{i\in I} 10^m + \sum_{i\in I}\sum_{j:v_i\in e_j} 10^{j-1} + \sum_{j\in J} 10^{j-1} \\ &= k\cdot 10^m + \left(\sum_{j\in J} 10^{j-1} + \sum_{j\notin J} 2\cdot 10^{j-1}\right) + \sum_{j\in J} 10^{j-1} \\ &= k\cdot 10^m + \sum_{j=1}^{m} 2\cdot 10^{j-1} = t\end{aligned}$$

כמבוקש.

בכיוון השני, נניח כי קיים פתרון לבעיית ה־SS. נסמן ב־I את קבוצת האינדקסים של ה־a_i־ים שנכללים בפתרון. לכל קבוצה $J \subseteq [m]$ של אינדקסים אם נתבונן בביטוי $\sum_{i\in I} a_i + \sum_{j\in J} b_j$ רק על סכומי החזקות עד ולא כולל חזקה t כלשהי, נקבל שסכום זה הוא לכל היותר

$$3\sum_{i=1}^{t-1} 10^t \leq 3\cdot\frac{10^t-1}{10-1} \leq 10^t - 1 < 10^t$$

ובמילים אחרות, אי אפשר להגיע אל אף חזקה של 10 על ידי חיבור של חזקות קטנות יותר מבין האיברים בקבוצה: הכרחי לחבר איברים שכוללים בהגדרתם את החזקות הללו של 10.

מכאן נסיק:

- החזקה $k\cdot 10^m$ מתקבל מהסכום $\sum_{i\in I} a_i$. מכיוון שבסכום זה מקבלים את החזקה $|I|\cdot 10^m$, נסיק ש־$|I| = k$.
- לכל $j < m$, החזקה $2\cdot 10^j$ מתקבלת מהסכום של b_{j+1} וחלק מאברי I. מכיוון ש־b_{j+1} תורם רק 10^j יחיד לסכום, עוד איבר מ־I חייב לתרום 10^j נוסף, כלומר הצומת v_i שמייצג אותו נוגע בקשת e_j, כמבוקש.

בכך מסתיימת הוכחת התקפות, ולכן ההוכחה כולה. ■

5.4.2 בעיית החלוקה Partition

הבעיה PARTITION מוגדרת באמצעות קבוצת מספרים טבעיים, שאנו מעוניינים לחלק לשתי תת־קבוצות זרות ומשלימות שסכומן זהה:

$$\text{PARTITION} = \left\{(a_1, a_2, \ldots, a_n) \mid \exists I\subseteq [n] : \sum_{i\in I} a_i = \sum_{i\notin I} a_i\right\}$$

$\text{PARTITION} \in \text{NP}$ עם היחס שנותן את I.

משפט 24.5 $\text{SS} \leq_p \text{PARTITION}$

כדי להבין את ההוכחה, ראשית נשים לב לכך ש־PARTITION היא מעין מקרה פרטי של SS. אם נסמן $A = \sum_{i=1}^{n} a_i$, אז PARTITION היא השאלה האם קיימת תת־קבוצה של המספרים שמסתכמת אל $\frac{A}{2}$. לכן, בהינתן בעיית SS עם סכום מבוקש k, אם נוסיף מספר איברים בצורה חכמה נוכל להפוך את הבעיה לבעיית SS שבה הסכום המבוקש הוא בדיוק חצי מסכום כל האיברים בקבוצה. **הוכחה:** בהינתן קלט $(a_1, a_2, \ldots, a_n, k)$ עבור SS, נסמן:

- $A = \sum_{i=1}^{n} a_i$
- $B = 2A - k$
- $C = A + k$

נוציא כפלט הרדוקציה את $(a_1,\dots,a_n,B,C)$.

בבירור הרדוקציה פולינומית כי החלק היחיד בביצוע שלה שתלוי באורך הקלט דורש רק את חישוב A, חישוב שהוא בבירור פולינומי.

נראה את תקפות הרדוקציה.

ראשית, נשים לב לכך שכעת סכום כל האיברים הוא

$$\sum_{i=1}^{n} a_i + B + C = A + B + C = A + (2A - k) + (A + k) = 4A$$

כלומר, קיים פתרון לבעיית ה־PARTITION אם ורק אם קיימת תת־קבוצה שמסתכמת אל $2A$.

בכיוון אחד, נניח ש־I הוא פתרון של בעיית ה־SS, כלומר $\sum_{a_i\in I} = k$. אז פתרון לבעיית ה־PARTITION יכלול את אברי I ועוד האיבר B; סכומם הוא בבירור $2A$.

בכיוון השני, נניח שקיים פתרון לבעיית ה־PARTITION. ראשית נשים לב לכך שבהכרח B,C אינם באותה תת־קבוצה, כי סכומם הוא $3A$ שגדול יותר מ־$2A$ המבוקש.

תהא I קבוצת האיברים שעם B בסכום - כפי שראינו, כולם איברים של הסדרה $a_1,a_2,\dots,a_n$ המקורית. מכיוון שסכום איברים אלו יחד עם B הוא $2A$ הרי שסכומם לאחר שמפחיתים ממנו את $B = 2A - k$ הוא k, כמבוקש. ■

5.4.3 בעיית החלוקה לתאים (Bin Packing)

בבעיית החלוקה לתאים BP, נתונים לנו איברים $a_1,\dots,a_n$ שהם מספרים טבעיים ובנוסף לכך נתונים לנו k תאים שקיבולת כל אחד מהם היא B. המטרה היא למצוא חלוקה של המספרים לתאים, כך שסכום המספרים בתא אינו עולה על B.

הבעיה בבירור ב־NP, וקיימת רדוקציה פשוטה $\text{PARTITION} \le_p \text{BP}$ שעל קלט $a_1,\dots,a_n$ מחזירה את אותה קבוצת איברים, $k = 2$ ו־$B = \frac{\sum_{i=1}^{n} a_i}{2}$.

5.4.4 בעיות של גרפים המילטוניים

אחת מהדוגמאות שבה פתחנו את הדיון על בעיות ב־NP הייתה זו של **גרפים המילטוניים**. גרפים הם המילטוניים אם קיים בהם מסלול שמבקר בכל צומת בדיוק פעם אחת. המסלול הזה עשוי להיות מעגל (להתחיל ולהסתיים באותה צומת; אנו לא מכלילים את נקודת הסיום בספירה) או לא; והגרף עשוי להיות מכוון או לא. זה מוביל לארבע שפות שונות:

- HC - שפת הגרפים **הלא מכוונים** שקיים בהם **מעגל** המילטוני.
- HL - שפת הגרפים **הלא מכוונים** שקיים בהם **מסלול** המילטוני.
- DHC - שפת הגרפים **המכוונים** שקיים בהם **מעגל** המילטוני.
- DHL - שפת הגרפים **המכוונים** שקיים בהם **מסלול** המילטוני.

כל ארבע השפות הללו שייכות בבירור ל־NP; בכולן היחס פשוט כולל את המסלול (שעשוי להיות מעגל), שהוא פשוט תמורה על כל צמתי הגרף. האתגר הוא להראות כי כל השפות הללו הן NP־שלמות, וזה מתבצע באמצעות שרשרת הרדוקציות הבאה:

$$\text{VC} \le_p \text{DHC} \le_p \text{HC} \le_p \text{HL} \le_p \text{DHL}$$

הרדוקציה הראשונה, $\text{VC} \le_p \text{DHC}$, היא הקשה בכולן; על כן נשמור אותה לסוף ונציג תחילה את הרדוקציות האחרות.

$\mathbf{DHC} \le_p \mathbf{HC}$ אנו רוצים לפתור את הבעיה של מציאת מעגל המילטוני בגרף **מכוון** באמצעות שימוש בגרף לא מכוון. אם ניקח גרף מכוון ופשוט נמחק את כיווני הקשתות, יוכלו להיווצר מעגלים חדשים שלא היו קיימים קודם, ואנו רוצים להימנע מכך.

דרך חכמה יותר (אבל שעדיין אינה עובדת) לשמור על האופי המכוון של הגרף היא לפצל כל צומת v בגרף המקורי לזוג צמתים v_{in}, v_{out} כך שקיימת בגרף החדש קשת $v_{in} \to v_{out}$, ואם בגרף המקורי הייתה קשת $u \to v$ אז בגרף החדש נוסיף קשת $v_{out} \to u_{in}$.

הבניה הזו עלולה להיכשל שכן מרגע שהמסלול נכנס אל v_{in} אין דרך לחייב אותו להמשיך משם אל v_{out}; הוא עשוי תחת זאת לעבור אל צומת לא קשור, u_{out} שמחובר אל v_{in}, ולבקר ב־v_{out} רק הרבה בהמשך. אנחנו חייבים להוסיף עוד משהו ש"יחייב" אותנו, מרגע שנכנסנו אל v_{in}, להמשיך אל v_{out}. את האפקט הזה ניתן להוסיף על ידי צומת נוסף באמצע ש"כלוא" בין v_{in} ו־v_{out}. אם לא נבקר בצומת הזה בהזדמנות הראשונה שלנו, "נשרוף" אחת משתי הקשתות שמחוברות אליו, מה שימנע מאיתנו להיכנס אליו בהמשך בלי להיתקע.

אם כן, הרדוקציה מוגדרת באופן פורמלי כך: בהינתן $G=(V,E)$ נחזיר $G'=(V',E')$ כך ש:

- $V'=\bigcup_{v\in V}\{v_{in},v_{middle},v_{out}\}$
- $E'=\bigcup_{v\in V}\{(v_{in},v_{middle}),(v_{middle},v_{out})\}\cup\{(v_{out},u_{in})\mid(v,u)\in E\}$

אם ב־G היה קיים המעגל ההמילטוני המכוון

$$v^1\to v^2\to\ldots\to v^n\to v^1$$

אז ב־G' יהיה קיים המעגל ההמילטוני הלא מכוון

$$v^1_{in}\to v^1_{middle}\to v^1_{out}\to v^2_{in}\to\ldots\to v^n_{out}\to v^1_{in}$$

מצד שני, אם ב־G' קיים מעגל המילטוני, הוא בהכרח מהצורה

$$v^1_{in}\to v^1_{middle}\to v^1_{out}\to v^2_{in}\to\ldots\to v^n_{out}\to v^1_{in}$$

או מאותה הצורה, אבל בכיוון ההפוך; במקרה זה, על ידי היפוך סדר הצמתים במעגל, נקבל שוב מעגל מהצורה לעיל, שממנו נובע קיומו ב־G של מעגל מהצורה

$$v^1\to v^2\to\ldots\to v^n\to v^1$$

$\mathbf{HC}\le_p\mathbf{HL}$ כעת אנו רוצים לפתור את הבעיה של קיום **מעגל** בגרף לא מכוון על ידי פתרון הבעיה של קיום **מסלול** בגרף לא מכוון. מעגל הוא בפרט מסלול, אבל קיימים מסלולים רבים שאינם מעגלים (למשל, בגרף "שרוך") כך שבהינתן G, אנו רוצים להרחיב אותו בדרך כלשהי לגרף G' שמסלול בו יתאים למעגל בגרף המקורי.

לצורך כך, נזכור כי למעגל אין נקודת התחלה או סיום ברורות; כל צומת בגרף יכול לשמש בתור נקודת ההתחלה והסיום של מעגל המילטוני. לכן נבחר צומת $v\in V$ שרירותי ב־G ונפצל אותו לשני צמתים, v_1,v_2, ששניהם מחוברים לאותם צמתים כמו v המקורי אבל לא מחוברים זה לזה.

בפני עצמו, הפיצול הזה לא יספיק לנו כי דבר לא מבטיח לנו ש־v_1,v_2 ישמשו בתור נקודות ההתחלה והסיום של המסלול. לכן נוסיף לגרף עוד שני צמתים מיוחדים: v_{start} ו־v_{end}, ונחבר אותם אל v_1,v_2 בהתאמה.

כלומר, הרדוקציה תוגדר פורמלית כך: בהינתן $G=(V,E)$ נחזיר $G'=(V',E')$ כך ש:

- $V'=(V\backslash\{v\})\cup\{v_1,v_2,v_{start},v_{end}\}$
- $E'=\bigcup_{(v,u)\in E}\{(v_1,u),(v_2,u)\}\cup\{(u,w)\mid u,w\in V\backslash\{v\}\}\cup\{(v_{start},v_1),(v_2,v_{end})\}$

בכיוון אחד, אם קיים ב־G מעגל המילטוני, נכתוב אותו כך ש־v הוא הצומת הראשון והאחרון:

$$v\to u_1\to u_2\to\ldots\to u_k\to v$$

וכעת ב־G' קיים המסלול ההמילטוני

$$v_{start} \to v_1 \to u_1 \to u_2 \to \ldots \to u_k \to v_2 \to v_{end}$$

בכיוון השני, נשים לב לכך שמכיוון שדרגת v_{start}, v_{end} היא 1, הופעה שלהם במסלול בגרף היא רק בתחילת או סיום המסלול; לא ניתן גם להיכנס וגם לצאת מהם. בהינתן מסלול המילטוני ב־G' נניח בלי הגבלת הכלליות שהוא מתחיל ב־v_{start} ומסתיים ב־v_{end} (אחרת אפשר להפוך את סדר כל הצמתים במסלול). מכיוון ש־v_{start} מחובר רק ל־v_1 ו־v_{end} מחובר רק ל־v_2, המסלול חייב להיות מהצורה

$$v_{start} \to v_1 \to u_1 \to u_2 \to \ldots \to u_k \to v_2 \to v_{end}$$

בפרט המסלול הזה מראה את קיום הקשתות (v, u_1) ו־(u_k, v) ב־G, כך שהמסלול הבא ב־G הוא חוקי:

$$v \to u_1 \to u_2 \to \ldots \to u_k \to v$$

מה שמסיים את הוכחת תקפות הרדוקציה.

$\mathbf{HL} \le_p \mathbf{DHL}$ כעת אנו מעוניינים לפתור את בעיית המסלול ההמילטוני בגרף **לא מכוון** בעזרת פתרון שלה עבור גרף **מכוון**. בהינתן גרף לא מכוון G, אם היינו מגדירים $G' = G$ הפלט שלנו היה "לא חוקי" שכן לא היו בו כיוונים לקשתות. אם סתם נכוון את קשתות G באופן אקראי, בהחלט ייתכן שנאבד מסלולים המילטוניים שקודם היו שם (אפילו בגרף שהוא כולו שרוך אם לא נכוון את כל הקשתות בכיוון המתאים נקבל גרף ללא מסלול המילטוני כלל).

הפתרון במקרה זה הוא פשוט ־ במקום כל קשת בגרף, להוסיף **שתי** קשתות, אחת לכל כיוון; זוהי טכניקה סטנדרטית למעבר מגרף לא מכוון לגרף מכוון.

כלומר, הרדוקציה תוגדר פורמלית כך: בהינתן $G = (V, E)$ נחזיר $G' = (V, E')$ כך ש:

- $E' = \bigcup_{(u,v)\in E} \{(u,v), (v,u)\}$

אם ב־G היה מסלול המילטוני

$$v_1 \to v_2 \to \ldots \to v_n$$

אותו מסלול יהיה קיים גם ב־G' שכן יש קשת $v_1 \to v_2$ ב־G אם ורק אם יש קשת $v_1 \to v_2$ ב־G'. באותו האופן גם מסלול ב־G' מתורגם למסלול זהה ב־G.

$\mathbf{VC} \le_p \mathbf{DHC}$ על מנת לפתור את בעיית הכיסוי בצמתים בעזרת מעגל המילטוני בגרף מכוון, נפעל כך: בהינתן גרף $G = (V, E)$ ו־k, נבנה גרף G' כך שלכל קשת $e \in E$ קיים רכיב ב־G' שמורכב מ**ארבעה** צמתים. אם $e = (v, u)$, כלומר הקשת e נוגעת בצמתים v, u אז נוסיף לגרף את הרכיב הבא:

$$\begin{array}{ccc} \downarrow & & \downarrow \\ [v,e,0] & \leftrightarrow & [u,e,0] \\ \downarrow & & \downarrow \\ [v,e,1] & \leftrightarrow & [u,e,1] \\ \downarrow & & \downarrow \end{array}$$

כלומר, ל־G' התווספו הצמתים מהצורה $[x, e, i]$ כאשר $x \in \{u, v\}$ ו־$i \in \{0, 1\}$, והקשתות

- $[x, e, 0] \to [x, e, 1]$ עבור $x \in \{u, v\}$

- $[v,e,i]\to[u,e,i]$ ו־$[u,e,i]\to[v,e,i]$ עבור $i\in\{0,1\}$

כפי שהאיור מרמז, נוסף על קשתות אלו יש לרכיב גם "כניסה" (לצמתים עם 0) ו"יציאה" (לצמתים עם 1); נראה מהיכן ולהיכן בקרוב.

הרעיון ברכיב זה הוא שמסלול שמגיע לאחד משני צמתי ה"כניסה" שלו יכול לעבור ממנו הישר אל צומת ה"יציאה" שאחריו, או לבקר בכל ארבעת צמתי הרכיב לפני שיצא מאותו מקום. הרעיון הוא שאם הכיסוי בצמתים שלנו יכיל את **שני** הצמתים שנוגעים ב־e, אז ניכנס אל הרכיב פעמיים, פעם אחת לכל צומת, ובמקרה זה נצא מייד; ואילו אם הכיסוי בצמתים שלנו מכיל רק אחד משני צמתים אלו אז נגיע לרכיב רק פעם אחת, דרך הכניסה שמתאימה לצומת זה, ואז נבקר בכל ארבעת צמתי הרכיב.

לכל צומת $v\in V$, יהיו $e_1,e_2,\ldots,e_m$ הקשתות שבהן v נוגע. נבנה ב־G' "שרשרת" של הצמתים של v ברכיבים שלו שמתאימים לקשתות אלו, באופן הבא:

$$\to[v,e_1,0]\to[v,e_1,1]\to[v,e_2,0]\to[v,e_2,1]\to\cdots\to[v,e_m,0]\to[v,e_m,1]\to$$

במילים אחרות, הוספנו לקשתות G' את כל הקשתות מהצורה

- $[v,e_{i+1},0]\to[v,e_i,1]$ עבור $1\le i<m$

האינטואיציה מאחורי ה"שרשרת" היא שאם v הוא אחד מהצמתים בכיסוי בצמתים שלנו, אז המעגל ההמילטוני שאנו בונים יעבור בכל השרשרת, ובכך ימחק את הצמתים של v ששייכים לכל קשת שנוגעת ב־v. עבור קשת e שאינה מכוסה על ידי צומת אחר מהכיסוי, בעת הכניסה ל־$[v,e,0]$ המסלול יעבור דרך זוג הצמתים ששייכים לצומת השני בו נוגעת e לפני שיעבור אל $[v,e,1]$ וימשיך במסעו על פני השרשרת.

טרם ציינו מהן נקודות הכניסה והיציאה מהשרשרת. לשם כך אנו מוסיפים k צמתי עזר, כאשר k הוא הגודל של הכיסוי בצמתים שנכלל בקלט (G,k) שממנו אנו מבצעים רדוקציה. נסמן את צמתי העזר ב־$a_1,\ldots,a_k$. הרעיון הוא שאחרי כל כניסה לצומת עזר, המעגל שלנו "בוחר" v כלשהו מהגרף והולך על גבי השרשרת שלו. לשם כך נוסיף לגרף את הקשתות הבאות:

- $a_i\to[v,e_1^v,0]$ לכל $1\le i\le k$ ו־$v\in V$
- $[v,e_m^v,1]\to a_i$ לכל $1\le i\le k$ ו־$v\in V$

כאשר e_1^v היא הקשת הראשונה בשרשרת המתאימה ל־v ו־e_m^v היא הקשת האחרונה בשרשרת זו.

הרדוקציה היא בבירור פולינומית (ב־G' מספר פולינומי של צמתים וקשתות ביחס ל־G) וכיוון אחד של התקפות שלה ברור: אם ב־G יש כיסוי בצמתים מגודל k, אז מעגל המילטוני ב־G' עובר בשרשראות שמתאימות לצמתי הכיסוי באופן שתיארנו.

בכיוון השני של הרדוקציה, נשים לב לכך שכל מעגל המילטוני ב־G' צריך לעבור בכל הצמתים $a_1,\ldots,a_k$. מכל צומת כזה, היציאה היחידה היא לצומת מהצורה $[v,e_1^v,0]$ המתאים לצומת $v\in V$; ניקח את כל הצמתים הללו לכיסוי בצמתים של G. כדי להיווכח בכך שזה כיסוי בצמתים, נשים לב לכך שמרגע הכניסה אל $[v,e_1^v,0]$ המסלול חייב להמשיך עם השרשרת עד סופה ב־$[v,e_m^v,1]$. גם אם ברכיב כלשהו הוא יבחר לעבור אל חלקו השני של הרכיב, הוא ייאלץ לחזור לחלקו הראשון כדי לצאת ממנו. אם כן, כל רכיב שמתאים לקשת e הופיע במסלול רק כחלק מהמעבר בשרשרת של אחד משני הצמתים שמחוברים אל e, ולכן אחד מהצמתים שמחוברים אל e שייך לכיסוי שבנינו.

6 נושאים נוספים

6.1 אלגוריתמי קירוב

6.1.1 הגדרה

עד כה העיסוק שלנו בבעיות NP־שלמות התמקד בבעיות הכרעה: בעיות כן/לא. במקרים רבים, הבעיות הללו קשורות בקשר הדוק לבעיות **אופטימיזציה**. למשל:

- בהינתן פסוק CNF φ, למצוא את המספר המקסימלי של פסוקיות של φ שניתן לספק.
- בהינתן גרף G, למצוא את הגודל המינימלי של כיסוי בצמתים עבור G.

- בהינתן מספרים $(a_1,\ldots,a_n)$ ו־B כך ש־$a_i \le B$, למצוא את המספר המינימלי k של תאים מגודל B שבהם ניתן לאכסן את המספרים.

פורמלית, אלו הן בעיות של **חישוב פונקציות**. כאשר הפונקציות הללו קשורות בקשר הדוק שכזה לבעיות NP־שלמות, חישוב יעיל שלהן הוא על פי רוב שקול לפתרון יעיל של הבעיה. נדגים זאת:

תהא f_{VC} הפונקציה כך ש־$f_{VC}(G)$ הוא ה־k המינימלי עבורו קיים כיסוי בצמתים ל־G (הפונקציה מוגדרת היטב לכל קלט שכן $k \le |V|$).

טענה 1.6 $\mathrm{VC} \in \mathrm{P} \iff f_{VC} \in \mathrm{POLY}$

הוכחה: בכיוון אחד, אם $f_{VC} \in \mathrm{POLY}$ אז בהינתן קלט (G,k) ל־VC, מכונה עבורו תבדוק האם $k \ge f_{VC}(G)$ ותקבל אם ורק אם אי־שוויון זה מתקיים. חישוב $f_{VC}(G)$ הוא פולינומי, על פי ההנחה שלנו, ולכן המכונה היא פולינומית.

בכיוון השני, אם $\mathrm{VC} \in \mathrm{P}$ אז P=NP ולכן זיהוי יעיל גורר חיפוש יעיל. נגדיר את היחס הבא:

$$S = \{((G,k),B) \mid B \text{ is vertex cover of } G \text{ of size } k\}$$

היחס בבירור ניתן לזיהוי יעיל כי בהינתן B קל לבדוק את גודלו וכי הוא מהווה כיסוי בצמתים של G; על כן הוא ניתן לחיפוש יעיל. כעת נפעל כך: נבצע חיפוש יעיל על הזוגות $(G,1),(G,2),(G,3),\ldots$ עד לפעם הראשונה בה יתקבל B; כאשר זה יקרה, נעצור ונוציא כפלט את גודלו של B. ■

מכיוון שפתרון בעיית האופטימיזציה אינו עומד על הפרק, אפשר לשאול שאלה אחרת ־ האם ניתן למצוא לה פתרון **מקורב**. אם לפנינו גרף בן 2000 צמתים והכיסוי בצמתים המינימלי שלו הוא מגודל 30, לא כל כך נורא אם נמצא בגרף כיסוי בצמתים מגודל 60 לכל היותר ־ פי 2 יותר מהמינימום "האמיתי". באופן מפתיע למדי, פתרון מקורב שכזה אכן אפשרי בזמן פולינומי, גם מבלי שהדבר יגרור $\mathrm{P} = \mathrm{NP}$.

ראשית נציג את ההגדרות המדוייקות למהו אלגוריתם קירוב.

הגדרה 2.6 תהא $f : \Sigma^* \to \mathbb{N}$ פונקציה ותהא A מ"ט פולינומית שמקבל קלט מ־Σ^* ומחזיר פלט ב־$\mathbb{N}$. יהיו $d, \alpha > 0$ ממשיים.

- נאמר ש־A היא d־קירוב **חיבורי** של f אם לכל $x \in \Sigma^*$ מתקיים $|A(x) - f(x)| \le d$, כלומר $f(x) - d \le A(x) \le f(x) + d$.
- נאמר ש־A היא α־קירוב **כפלי** של f אם לכל $x \in \Sigma^*$ מתקיים $\frac{1}{\alpha} f(x) \le A(x) \le \alpha f(x)$.
- אם f מייצגת בעיית **מינימיזציה** נאמר ש־A היא α־קירוב **כפלי** של f אם לכל $x \in \Sigma^*$ מתקיים $f(x) \le A(x) \le \alpha f(x)$.
- אם f מייצגת בעיית **מקסימיזציה** נאמר ש־A היא α־קירוב **כפלי** של f אם לכל $x \in \Sigma^*$ מתקיים $\alpha f(x) \le A(x) \le f(x)$.

6.1.2 אלגוריתמי קירוב קונקרטיים

אלגוריתם קירוב ל־VC ראשית נציג את האלגוריתם שהבטחנו ־ אלגוריתם 2־קירוב כפלי לבעיית VC. האלגוריתם מתבסס על המושג של **שידוך**:

הגדרה 3.6 שידוך בגרף G הוא תת־קבוצה $M \subseteq E$ כך שאין שתי קשתות ב־M עם צומת משותף. שידוך הוא **מקסימלי** אם לא ניתן להוסיף לו קשת שאין לה צומת משותף עם קשתות אחרות בשידוך.

טענה 4.6 אם שידוך M הוא מקסימלי אז קבוצת הצמתים $V_M = \{v \in V \mid \exists e \in M : v \in e\}$ היא כיסוי בצמתים של G.

הוכחה: תהא $e \in E$ כלשהי. אם $e \in M$ אז על פי הגדרה, ב־V_M יש קשת שנוגעת ב־e. אם $e \notin M$ אז מכך ש־M מקסימלי נובע של־e יש צומת משותף עם אחת מקשתות M, ולכן צומת זה שייך ל־V_M. ■

אם כן, מציאת שידוך מקסימלי בגרף מאפשרת לנו למצוא כיסוי בצמתים שלו; אך נשאלת השאלה עד כמה כיסוי בצמתים זה רחוק מלהיות אופטימלי.

משפט 5.6 תהא f_{VC} הפונקציה אשר מחזירה עבור גרף את הגודל המינימלי של כיסוי בצמתים עבורו. יהא G גרף ו־M שידוך מקסימלי בגרף, אז $f_{\text{VC}}(G) \le |V_M| \le 2f_{\text{VC}}(G)$.

הוכחה: יהא B כיסוי בצמתים מגודל מינימלי של G, כלומר $|B| = f_{\text{VC}}(G)$. מכיוון ש־V_M הוא כיסוי בצמתים ו־B מגודל מינימלי, אז $|B| \le |V_M|$.

לכל קשת $e \in M$, לפחות אחד משני הצמתים שלה שייך ל־B. מכיוון ש־M שידוך, ההתאמה שמחזירה לכל $e \in M$ צומת של e ששייך ל־B היא חח"ע, אחרת היו לנו שתי קשתות ב־M שיש להן צומת משותף. כלומר, $|M| \le |B|$. מכיוון ש־$|V_M| = 2|M|$ נקבל $|V_M| \le 2|B|$, כמבוקש. ■

מכאן שאלגוריתם 2־קירוב עבור VC פשוט צריך למצוא שידוך מקסימלי, ושידוך שכזה קל למצוא בזמן פולינומי בעזרת אלגוריתם חמדני פשוט.

אלגוריתם למציאת שידוך מקסימלי בגרף G:

- אתחול: $M = \emptyset$.
- לכל $e \in E$:
  - אם ל־e אין צומת משותף עם אף קשת ב־M, $M \leftarrow M \cup \{e\}$
- פלט: M

האלגוריתם פולינומי שכן הוא עובר פעם אחת על E ולכל פעם כזו עובר על כל אברי M, כלומר הוא $O\left(|E|^2\right)$.

נכונות האלגוריתם נובעת מכך שבסיומו, כל קשת $e \in E$ היא או שייכת אל M, או בעלת צומת משותף עם קשת ב־M.

אלגוריתם קירוב ל־BP נעבור כעת לבעיית האופטימיזציה המתאימה עבור BP: בהינתן מספרים טבעיים $a_1, \ldots, a_n$ וגודל תא B כך ש־$a_i \le B$ לכל $1 \le i \le n$, המטרה היא למצוא את מספר התאים **המינימלי** k כך שניתן לשכן את כל ה־a_i־ים ב־k התאים.

גם כאן האלגוריתם החמדני ישיג לנו 2־קירוב כפלי, במקרה זה אלגוריתם חמדני שעבור כל איבר חדש, מחפש לו מקום בתא קיים ואם אין מקום - פותח בשבילו תא חדש.

- אתחול: $k = 1, C_1 = 0$ (C_j הוא סכום הערכים בתא j)
- לכל x_i ב־$\{x_1, x_2, \ldots, x_n\}$:
  - אם קיים C_j עבור $1 \le j \le k$ כך ש־$C_j + x_i \le B$, הציבו $C_j \leftarrow C_j + x_i$
  - אחרת, הציבו $C_{k+1} \leftarrow x_i, k \leftarrow k+1$
- פלט: k.

האלגוריתם בבירור פולינומי. נותר להוכיח כי הוא מהווה 2־קירוב. לשם כך נוכיח את הטענה הבאה: בכל שלב של ריצת האלגוריתם, כל התאים למעט אולי אחד מלאים לפחות במחציתם (כלומר, מכילים לפחות $\frac{B}{2}$).

נוכיח את הטענה באינדוקציה על x_i. הבסיס עבור x_1 ברור כי בשלב זה קיים רק תא אחד. נניח כי הטענה הייתה נכונה עד x_i ונוכיח עבור x_{i+1}. על פי הנחת האינדוקציה כל התאים למעט אולי אחד שנסמן ב־C_j מלאים לפחות עד כדי מחציתם. כעת יכול לקרות אחד משני דברים:

1. יתווסף תא חדש; זה קורה במקרה של־x_{i+1} אין מקום באף תא פנוי. בפרט אין לו מקום ב־C_j. אם $C_j < \frac{B}{2}$ נסיק ש־$x_{i+1} \ge \frac{B}{2}$ ולכן התא החדש שאליו מוסיפים את x_{i+1} יהיה מלא לפחות עד כדי מחציתו. אם לעומת זאת $C_j \ge \frac{B}{2}$ הרי שכל התאים לפני הוספת x_{i+1} היו מלאים לפחות עד כדי מחציתם, ולכן לאחר הוספת התא החדש כל התאים למעט אולי אחד (החדש) מלאים לפחות עד כדי מחציתם.

2. לא יתווסף תא חדש; במקרה זה, התכונה "כל התאים למעט אולי אחד מלאים לפחות במחציתם" משתמרת (כי רק נוסיף איבר לאחד מהתאים ובכך נמלא אותו עוד יותר).

כעת נוכיח כי האלגוריתם הוא 2־קירוב. יהיו $a_1, \ldots, a_n$ ו־B כלשהם, ויהא k הערך שאלגוריתם הקירוב החזיר על קלט זה, ו־k^* הערך האופטימלי האמיתי.

מכיוון שהקיבולת המקסימלית של k^* תאים היא $B \cdot k^*$ אנחנו יודעים שמתקיים $\sum_{i=1}^{n} a_i \le B \cdot k^*$.

מצד שני, כאשר אלגוריתם הקירוב מסיים את ריצתו, כל התאים למעט אחד מלאים לפחות במחציתם, כלומר $\sum_{i=1}^{n} a_i > \frac{B}{2} \cdot (k-1)$. אי השוויון הוא חזק כי התא הנוסף, שלא ספרנו, אינו ריק.
משני אי השוויונים נקבל:

$$\frac{B}{2}(k-1) < B \cdot k^*$$
$$\Downarrow$$
$$\frac{k-1}{2} < k^*$$
$$\Downarrow$$
$$k-1 < 2k^*$$
$$\Downarrow$$
$$k \le 2k^*$$

כמבוקש.

6.1.3 קושי לקירוב של בעיות

במובן מסויים, כל הבעיות ה־NP־שלמות דומות זו לזו; כל אחת ניתנת לרדוקציה אל האחרת, ואם אחת שייכת אל P, כולן שייכות אל P. מצד שני, יש ביניהן הבדלים מהותיים שאחד מהם בא לידי ביטוי בכך שלחלק מהבעיות יש אלגוריתמי קירוב יעילים ולאחרות אין.
נציג מספר דוגמאות לטענה זו.

הבעיה #SAT הפונקציה #SAT מוגדרת בתור מספר ההשמות המספקות של פסוק φ. זוהי דוגמא קלאסית ל**בעיית ספירה** (לבעיות ספירה יש מחלקות סיבוכיות משל עצמן עם תוצאות מעניינות ולא טריוויאליות, אך לא נציג אותן כאן).
ברור שחישוב מדויק של #SAT מאפשר להכריע את SAT: בהינתן φ, מחשבים את #SAT על הפסוק ועונים "כן" אם ורק אם התוצאה שונה מ־0.

טענה 6.6 אם קיים α־קירוב כפלי ל־#SAT אז P=NP.

הוכחה: נניח שקיים $\alpha > 0$ וקיימת מ"ט פולינומית A כך ש־

$$\frac{\#\mathrm{SAT}(\varphi)}{\alpha} \le A(\varphi) \le \alpha\#\mathrm{SAT}(\varphi)$$

בהינתן פסוק φ, מ"ט שמכריעה האם $\varphi \in \mathrm{SAT}$ תפעל כך: תחשב את $A(\varphi)$ ותקבל אם ורק אם $A(\varphi) \neq 0$.
פולינומיות המכונה ברורה. נוכיח את נכונותה:
אם $\varphi \in \mathrm{SAT}$ אז $\#\mathrm{SAT}(\varphi) \ge 1$ ולכן $A(\varphi) \ge \frac{\#\mathrm{SAT}(\varphi)}{\alpha} \ge \frac{1}{\alpha} > 0$ ולכן המכונה שבנינו תקבל.
אם $\varphi \notin \mathrm{SAT}$ אז $\#\mathrm{SAT}(\varphi) = 0$ ולכן

$$0 = \frac{\#\mathrm{SAT}(\varphi)}{\alpha} \le A(\varphi) \le \alpha\#\mathrm{SAT}(\varphi) \le 0$$

כלומר $A(\varphi) = 0$ ולכן המכונה שבנינו תדחה, כנדרש. ■

הוכחנו שלא קיים קירוב כפלי ל־$\#\mathrm{SAT}(\varphi)$ בהנחה ש־$\mathrm{P} \neq \mathrm{NP}$, אבל מה בדבר קירוב חיבורי? בהוכחה הקודמת הסתמכנו על כך שההפרדה בין $\#\mathrm{SAT}(\varphi) = 0$ ו־$\#\mathrm{SAT}(\varphi) = 1$ היא קלה יחסית לביצוע בעזרת קירוב כפלי, וההפרדה הזו מספיקה כדי להכריע את SAT. אולם כבר קירוב 1־חיבורי ל־$\#\mathrm{SAT}(\varphi)$ יכול "לערבב" כך בין שני המקרים (להחזיר 1 כשהערך האמיתי הוא 0 ולהיפך). כך שנראה שהסיטואציה מורכבת יותר. אכן, נזדקק לתעלול נוסף כדי להוכיח שקיים קושי קירוב גם במקרה הזה - "ניפוח" מלאכותי של מספר ההשמות המספקות של φ.

טענה 7.6 אם קיים d־קירוב חיבורי ל־$\#\text{SAT}$ אז $\text{P}=\text{NP}$.

הוכחה: נניח כי קיימת מ"ט פולינומית A שהיא d־קירוב של $\#\text{SAT}$. אז נבנה מ"ט שמכריעה את SAT ופועלת באופן הבא על קלט φ:

ראשית, המכונה בונה פסוק חדש $\varphi' = \varphi \wedge (y_1 \vee \neg y_1) \wedge (y_2 \vee \neg y_2) \wedge \ldots \wedge (y_k \vee \neg y_k)$ שמתקבל מהוספת k פסוקיות אל φ על משתנים **חדשים** $y_1, \ldots, y_k$, כך שכל פסוקית היא טאוטולוגיה, כלומר מסתפקת תחת כל השמה למשתנים שלה. את הערך המדויק של k נקבע בהמשך, בהתאם למה שמתאים לנו.

אם τ היא השמה מספקת של φ, כל בחירה של השמת ערכים למשתנים $y_1, \ldots, y_k$ מרחיבה את τ להשמה מספקת τ' של φ'. כלומר, לכל השמה מספקת של φ קיימות 2^k השמות מספקות של φ', כלומר $\#\text{SAT}(\varphi') = 2^k \#\text{SAT}(\varphi)$.

נבחר את k כך ש־$2d < 2^k$; קונקרטית, אפשר לבחור $k = \lceil \lg d \rceil + 1$. נשים לב לכך ש־$k$ הוא **קבוע**; הוא אינו תלוי באורך φ כלל, ולכן הוספת k הפסוקיות לוקחת זמן פולינומי ומגדילה את φ פולינומית.

כעת, נריץ את A על φ' ונדחה אם ורק אם $A(\varphi') \le d$.

נוכיח את נכונות הבניה.

מצד אחד, אם φ אינו ספיק אז $\#\text{SAT}(\varphi) = 0$ ולכן $\#\text{SAT}(\varphi') = 2^k \#\text{SAT}(\varphi) = 0$. לכן $A(\varphi') \le \#\text{SAT}(\varphi') + d \le d$ כך שבמקרה זה אנו אכן דוחים כנדרש.

אם φ ספיק אז $\#\text{SAT}(\varphi) \ge 1$ ולכן

$$\#\text{SAT}(\varphi') = 2^k \#\text{SAT}(\varphi) \ge 2^k > 2d$$

ולכן

$$\begin{aligned} A(\varphi') &\ge \#\text{SAT}(\varphi') - d \\ &> 2d - d \ge d \end{aligned}$$

כלומר, $A(\varphi') > d$ ולכן במקרה זה נקבל, כנדרש. ■

הבעיה MAX-3SAT נסיים את הדיון על אלגוריתמי קירוב עם דוגמא מורכבת מעט יותר, שמובילה אותנו אל אחד מהמשפטים המפורסמים בתורת הסיבוכיות ־ משפט ה־PCP, שלא נציג בצורה מלאה כאן.

נתחיל עם בעיה תמימה למראה: MAX-3SAT. בבעיה זו נתון פסוק 3CNF φ ויש למצוא את המספר המקסימלי של פסוקיות של φ שניתן לספק בו זמנית. כמובן שחישוב יעיל של MAX-3SAT יוביל להכרעת φ ־ פשוט נחשב את הערך ונבדוק אם הוא שווה למספר הפסוקיות של φ.

מה בדבר אלגוריתם קירוב? כדי לפשט את ניתוח הבעיה, נניח שכל פסוקית של φ כוללת שלושה משתנים **שונים** זה מזה. בהינתן הנחה זו, ניתן להראות כי לפחות $\frac{7}{8}$ מהפסוקיות של φ הן ספיקות בו זמנית. נראה זאת באמצעות טכניקה המכונה **השיטה ההסתברותית** שמאפשרת להשתמש בכלים מתורת ההסתברות כדי להוכיח תוצאות לא הסתברותיות.

יהא $\varphi = C_1 \wedge \ldots \wedge C_m$ פסוק 3CNF בעל m פסוקיות שבכל אחת מהן כל המשתנים שונים זה מזה, ויהיו $x_1, \ldots, x_n$ משתני φ. נגדיר מרחב הסתברות אחיד על ההשמות האפשריות ל־$x_1, \ldots, x_n$, כלומר כל אחת מ־2^n ההשמות האפשריות ל־φ נבחרת בהסתברות זהה, ובפרט לכל משתנה x_i ההסתברות לקבל T שווה להסתברות לקבל F והיא $\frac{1}{2}$.

תהא $C_j = (l_1 \vee l_2 \vee l_3)$ פסוקית כלשהי של φ. קיימות בדיוק 8 השמות למשתנים המופיעים בה (שכן הנחנו כי כל המשתנים הללו שונים זה מזה) ובדיוק אחת מהן לא תספק את הפסוקית (זו שבה כל הליטרלים מקבלים F) ויתר 7 ההשמות כן יספקו את הפסוק. מכאן קל להסיק שההסתברות של השמה אקראית לספק את C_j היא $\frac{7}{8}$.

נגדיר כעת משתנה מקרי X_j שהוא אינדיקטור של "הפסוקית C_j מסתפקת". דהיינו, הוא מחזיר 1 על השמות שמספקות את C_j ו־0 על השמות שאינן מספקות את C_j. מכאן שהתוחלת שלו שווה להסתברות שיקבל 1, כלומר $\text{E}[X_j] = \frac{7}{8}$.

כעת, המשתנה המקרי $X = \sum_{j=1}^{m} X_j$ סופר את הפסוקיות של φ שהסתפקו תחת השמה אקראית. נשתמש כעת בתוצאה בסיסית מתורת ההסתברות ־ **לינאריות התוחלת** ־ ונקבל:

$$\text{E}[X] = \text{E}\left[\sum_{j=1}^{m} X_j\right] = \sum_{j=1}^{m} \text{E}[X_j] = \sum_{j=1}^{m} \frac{7}{8} = \frac{7}{8} m$$

כעת נעבור מתוצאה "הסתברותית" אל תוצאה דטרמיניסטית: מכיוון שתוחלת X היא $\frac{7}{8}m$, המסקנה היא שקיימת השמה אחת לפחות שעבורה X מקבל לכל הפחות את הערך הזה, אחרת תוחלת X הייתה בהכרח נמוכה יותר. כלומר, קיימת השמה ל־φ שמספקת לפחות $\frac{7}{8}m$ מהפסוקיות, כמבוקש.

תוצאה זאת מצביעה על קיום אלגוריתם קירוב $\frac{7}{8}$-כפלי לבעיית MAX-3SAT: בהינתן φ בעל m פסוקיות, האלגוריתם A מוציא כפלט $\frac{7}{8}m$. כעת, אם $f(\varphi)$ הוא מספר הפסוקיות שניתן לספק בו זמנית ב־φ, אז $A(\varphi)=\frac{7}{8}m\le f(\varphi)$ על פי התוצאה שראינו, ומצד שני $f(\varphi)\le m$, ולכן $\frac{7}{8}f(\varphi)\le\frac{7}{8}m=A(\varphi)$. נקבל:

$$\frac{7}{8}f(\varphi)\le A(\varphi)\le f(\varphi)$$

כך ש־A הוא אכן אלגוריתם קירוב $\frac{7}{8}$-כפלי.

האם קיים אלגוריתם קירוב כפלי טוב יותר? יהא $0<\varepsilon$, אז ניתן להוכיח כי קיום אלגוריתם קירוב $\frac{7}{8}+\varepsilon$-כפלי ל־MAX-3SAT יוכיח כי $\mathrm{P}=\mathrm{NP}$. תוצאה זו נובעת מ**משפט ה־PCP** שלא נתאר במלואו כאן אך שימוש מרכזי שלו הוא להוכחות קושי דומות של אלגוריתמי קירוב.

6.2 הוכחה בלכסון לקיום שפה $L\in\mathrm{R}\backslash\mathrm{P}$

בחלקו הראשון של הקורס, שעסק בתורת החישוביות, אחת מהתוצאות המרכזיות שלנו היה קיומן של שפות $L\in\mathrm{RE}\backslash\mathrm{R}$, כדוגמת השפה HP. כפי שראינו, אין לנו תוצאה מקבילה עבור $\mathrm{NP}\backslash\mathrm{P}$ אף שראינו מועמדות רבות להיות שפות כאלו (כל השפות ה־NP־שלמות). עם זאת, אין זה אומר שאיננו יודעים להוכיח קיום של שפות שאינן ב־P אך שייכות למחלקה "קלה" יותר מ־RE. נדגים זאת עבור המחלקה R, אף שניתן להוכיח קיום של שפות שאינן ב־P ששייכות גם למחלקות קטנות יותר מ־R.

טכניקת ההוכחה שבה נשתמש תהיה **לכסון**. הרעיון בלכסון הוא לבנות מכונת טיורינג M כך שמובטח לנו ש־$L(M)\notin\mathrm{P}$ על ידי כך שלכל מכונת טיורינג פולינומית M' יהיה קיים קלט w כך ש־M,M' מחזירות תוצאות שונות על אותו הקלט.

משפט 8.6 קיימת $L\in\mathrm{R}\backslash\mathrm{P}$

הוכחה: נבנה מ"ט M שפועלת באופן הבא על קלט w:

1. אם w אינה מהצורה $w=\left(\langle M'\rangle,1^k\right)$ עבור מ"ט M' ו־$k\in\mathbb{N}$, M דוחה מייד.
2. M מריצה את M' על w במשך 2^k צעדים.
3. אם M' עצרה על w, M עונה הפוך ממנה.
4. אחרת, M עוצרת ודוחה.

מהגדרת M ברור כי היא עוצרת על כל קלט, כך ש־$L(M)\in\mathrm{R}$.

נוכיח כי $L(M)\notin\mathrm{P}$ (נשים לב לכך שגם אם M אינה פולינומית, זה לכשעצמו לא שולל את היתכנות הקיום של מ"ט פולינומית אחרת עבור אותה שפה).

תהא M' מ"ט פולינומית כלשהי. יהא $p(x)$ הפולינום שחוסם את זמן ריצתה לכל w.

מכיוון ש־p הוא פולינום, קיים מספר טבעי n כך ש־$p(n+|\langle M'\rangle|)<2^n$.

נתבונן בריצת M על הקלט $w=(\langle M'\rangle,1^n)$: על קלט זה, M תריץ את M' על w למשך 2^n צעדים. מכיוון ש־$p(|w|)=p(n+|\langle M'\rangle|)<2^n$, הרי ש־$M'$ תסיים את ריצתה במהלך אותם 2^n צעדים, ו־M תענה הפוך ממנה. אם כן, על הקלט w, M ו־M' אינן מסכימות, כך ש־$L(M)\neq L(M')$. מכיוון שזה המצב לכל מ"ט פולינומית, $L(M)\notin\mathrm{P}$. ■

6.3 סיבוכיות זיכרון

6.3.1 הגדרה ותוצאות בסיסיות

המשאב המרכזי שדנו בו בקורס היה **זמן החישוב** שמדדנו באמצעות מספר הצעדים שמבצעת מכונת טיורינג. מדד חשוב נוסף הוא **כמות הזיכרון** שבה החישוב משתמש. לכאורה, המודל של מכונת טיורינג משתמש ב"אינסוף" זיכרון, שהרי הסרט שעליו כותבת המכונה הוא אינסופי, אך בפועל ניתן לדרוש שהמכונה לא תגיע לתא ספציפי כלשהו, שמהווה את חסם הזיכרון של המכונה.

הגדרה 9.6 מכונת טיורינג חד סרטית M פועלת על קלט w **בסיבוכיות זיכרון** k אם הראש הקורא של המכונה אינו מגיע אל מימין לתא מספר k. בהינתן פונקציה $s:\mathbb{N}\to\mathbb{N}$ נאמר ש־M פועלת בסיבוכיות זיכרון s אם לכל $w\in\Sigma^{*}$, M פועלת על x בסיבוכיות זיכרון $s\left(|w|\right)$.

לפעמים אנו נדרשים להגדרה עדינה יותר. נאמר שאנו רוצים לעסוק במכונה שכמות הזיכרון שבה היא משתמשת היא קטנה יחסית לגודל הקלט ־ למשל $O\left(\log n\right)$ או אפילו $O\left(1\right)$. קיימים אלגוריתמים רבים שפועלים בסיבוכיות זיכרון שכזו, למשל אלגוריתמים של חיפוש. עם זאת, על פי ההגדרה שהצגנו, אם M רוצה לקרוא את כל הקלט w, אפילו מבלי שתכתוב שום דבר על הסרט בשום שלב, הראש שלה יגיע אל תא מס' $|w|$ כך שהיא תפעל בסיבוכיות זכרון $\Omega\left(n\right)$. בדומה, אם מכונה רוצה לכתוב פלט, סיבוכית הזיכרון תהיה לפחות אורך הפלט הזה.

על כן משתמשים לעתים בהגדרה שבה המכונה היא **תלת־סרטית**. הסרט הראשון, שעליו נכתב הקלט, הוא סרט **לקריאה בלבד** והמכונה לא יכולה לכתוב עליו דבר. הסרט השני הוא סרט **לכתיבה בלבד** והמכונה לא יכולה לקרוא את תוכן התאים בו לאחר שנכתבו (בכל פעם שהמכונה כותבת משהו בתא, היא נעה ימינה ואינה יכולה לנוע שמאלה שוב). הסרט השלישי הוא **סרט עבודה** והמכונה יכולה לקרוא ולכתוב בו כרצוננו. סיבוכיות הזיכרון של המכונה נמדדת לפי המיקום הימני ביותר שהראש מגיע אליו בסרט העבודה. ניתן גם לאפשר מספר סופי כלשהו של סרטי עבודה, וסיבוכיות הזיכרון תהיה סכום המקומות שאליהם הראשים הגיעו בכל הסרטים הללו.

אנו לא נזדקק להגדרות הללו שכן נעסוק במחלקת סיבוכיות שעבורה $O\left(n\right)$ הוא זיכרון קטן יחסית:

הגדרה 10.6 המחלקה PSPACE כוללת את כל השפות L כך שקיימת מכונת טיורינג M עם סיבוכיות זיכרון פולינומית עבורה $L\left(M\right)=L$.

באופן לא מפתיע במיוחד, סיבוכיות זמן גוררת סיבוכיות זיכרון:

טענה 11.6 אם $L\in\mathrm{P}$ אז $L\in\mathrm{PSPACE}$.

הוכחה: תהא M מ"ט בעלת חסם סיבוכיות זמן $p\left(n\right)$ כך ש־$L\left(M\right)=L$. אז בפרט M בעלת חסם סיבוכיות זיכרון $O\left(p\left(n\right)\right)$ כי גם אם בכל $p\left(n\right)$ הצעדים שלה M תבצע צעד ימינה, היא לא תעבור את התא $p\left(n\right)+1$ ■

באופן קצת יותר מפתיע, גם NP נכללת ב־PSPACE:

משפט 12.6 $\mathrm{NP}\subseteq\mathrm{PSPACE}$

הוכחה: תהא $L\in\mathrm{NP}$ ויהא R יחס חסום פולינומית על ידי פולינום $p\left(n\right)$ וניתן לזיהוי פולינומי כך ש־$L=\left\{x\mid\exists y:\left(x,y\right)\in R\right\}$. נבנה מכונה M לזיהוי L שפועלת כך: בהינתן x, לכל y כך ש־$|y|\le|p\left(x\right)|$ המכונה תבדוק האם $\left(x,y\right)\in L$. הבדיקה הזו דורשת זמן פולינומי ולכן זיכרון פולינומי; לאחר סיום הבדיקה ומעבר אל ה־y הבא ניתן **להשתמש שוב** באותו מקום שבו התבצע הבדיקה הקודמת, כך שגודל הזיכרון הנדרש עבור בדיקת **כל** ה־y האפשריים הוא פולינומי. ■

ההוכחה שלעיל ממחישה את ההבדל הגדול ביותר בין סיבוכיות זמן וסיבוכיות זיכרון: זיכרון ניתן למחזר, זמן לא. זה מוביל לכך ש־P היא מחלקה קטנה יחסית בעולם הגדול של תורת הסיבוכיות, ולעומת זאת PSPACE היא מחלקה גדולה למדי; מחלקות רבות שצצות בתורת הסיבוכיות (מחלקות לחישוב הסתברותי, "ההירככיה הפולינומית" שמתאימה לחישוב עם אורקלים, מערכות הוכחה אינטראקטיביות ועוד) כולן נכללות בה. לנוכח הפער הגדול הזה בין האופן שבו נתפסים גדלי P ו־PSPACE היה ניתן לקוות שקיימת הוכחה ש־$\mathrm{P}\ne\mathrm{PSPACE}$, אך אין כזו; השאלה האם $\mathrm{P}=\mathrm{PSPACE}$ היא שאלה פתוחה בדיוק כמו שאלת $\mathrm{P}=\mathrm{NP}$.

6.3.2 גרף הקונפיגורציות של מכונה

אם קיימת M כך ש־$L\left(M\right)=L$ ויש ל־M חסם סיבוכיות זיכרון $s\left(n\right)$, האם ניתן להניח ש־M **מכריעה** את L, כלומר עוצרת לכל קלט? התשובה שלילית ־ חסם על סיבוכיות זיכרון אינו מונע מהמכונה לרוץ לנצח (חשבו על מכונה שעושה אינסוף צעדי S). עם זאת, **ניתן** תמיד לבנות מכונה שמכריעה את L על ידי הרצה חכמה של M. אם ידוע לנו חסם סיבוכיות הזיכרון של M, אנחנו יכולים להסיק ממנו חסם על מספר הקונפיגורציות האפשריות של M, ואם M ביצעה מספר צעדים גדול יותר ממספר הקונפיגורציות האפשריות שלה, המסקנה היא ש־M הייתה פעמים באותה קונפיגורציה ־ כלומר היא בלולאה אינסופית ולא תעצור לעולם.

כדי לראות זאת פורמלית, ניעזר במושג של **גרף קונפיגורציות**:

הגדרה 13.6 בהינתן M,x כך ש־M יש חסם סיבוכיות זיכרון $s\left(n\right)$, **גרף הקונפיגורציות** של ריצת M על x הוא גרף שצמתיו הם כל הקונפיגורציות האפשריות של M שבהן המכונה משתמשת לכל היותר ב־$s\left(|x|\right)$ תאים מהסרט וקיימת קשת מכוונת מהצומת C אל הצומת C' אם C' היא הקונפיגורציה העוקבת של C.

נשים לב לכך שגרף הקונפיגורציות תלוי ב־s (חסם סיבוכיות גדול יותר עבור אותה מכונה יניב גרף קונפיגורציות גדול יותר) אבל אנו מאפשרים גם לקונפיגורציות שלא יופיעו בריצת M על x להופיע בו; מבחינתנו הגרף כלל את כל הקונפיגורציות **הפוטנציאליות** בריצת M על x, כשהמגבלה היחידה היא על אורך תוכן הסרט שבו המכונה השתמשה.

כזכור ממשפט קוק, אפשר לייצג קונפיגורציה מאורך k באמצעות מחרוזת ב־$\left(\Gamma\cup\Gamma\times Q\right)^{k}$, כלומר באמצעות $O\left(k\right)$ ביטים. מכאן שעבור מכונה עם חסם סיבוכיות זיכרון $s\left(n\right)$, גודל כל קונפיגורציה בביטים חסום על ידי $O\left(s\left(n\right)\right)$. אם מספר הביטים המקסימלי לייצוג קונפיגורציה הוא L, אז מספר הקונפיגורציות הכוללה האפשרי הוא 2^{L}.

אם נתבונן על מסלול בגרף הקונפיגורציות שאורכו הוא לפחות 2^{L}, אז במסלול זה מופיעים $2^{L}+1$ צמתים (הצומת ההתחלתי + הצומת שהגענו אליו אחרי כל צעד). מכיוון שבגרף יש רק 2^{L} צמתים, מעיקרון שובך היונים נובע שיש צומת שהופיע פעמיים על המסלול. מכיוון ש־M היא מכונה דטרמיניסטית, נובע מכך שכל הגעה לצומת הזה גוררת הגעה אינסופית אליו, ולכן אי־עצירה של המכונה (רעיון דומה מופיע בקורס באוטומטים ושפות פורמליות כאשר מוכיחים את **למת הניפוח**).

מכאן נסיק:

משפט 14.6 אם M מכונה בעלת סיבוכיות זיכרון פולינומית $p\left(n\right)$ אז קיימת מ"ט M' בעלת סיבוכיות זמן $O\left(2^{p(n)}\right)$ כך ש־$L\left(M'\right)=L\left(M\right)$.

נהוג לסמן ב־$\mathrm{EXPTIME}$ את מחלקת השפות שיש להן מכונת טיורינג בעלת סיבוכיות זמן $O\left(2^{p(n)}\right)$ עבור פולינום p כלשהו, כך שראינו כאן כי $\mathrm{PSPACE}\subseteq\mathrm{EXPTIME}$.

מדוע לא הצגנו כאן תוצאה כללית יותר, למשל **שלכל** חסם סיבוכיות זיכרון $s\left(n\right)$ קיימת מכונה שמכריעה את השפה בסיבוכיות זמן $O\left(2^{s(n)}\right)$? הסיבה לכך היא שלפעמים s עלולה להיות פונקציה מסובכת ואפילו לא ניתנת לחישוב, אבל חישוב $s\left(n\right)$ (או חסם עליה) הוא קריטי לצורך הטכניקה שהצגנו. אם כן, כאשר עוסקים פורמלית בסיבוכיות זיכרון נזקקים להגדרות מדויקות עוד יותר עבור סיטואציות כאלו.

6.3.3 השפה TQBF ומחלקת השפות ה־PSPACE־שלמות

אפשר לחשוב על בעיות NP בתור סוג של "משחק" לשחקן יחיד: בהינתן שפה $L\in\mathrm{NP}$ קיים לה יחס R_{L}, ואפשר לחשוב ה"משחק" שבו בהינתן x מטרת השחקן היא למצוא y כך ש־$\left(x,y\right)\in R_{L}$. למשל, בהינתן לוח סודוקו x, מטרת השחקן היא למצוא מילוי חוקי y של הלוח; בהינתן אוסף x של צורות, מטרת השחקן היא למצוא דרך y לסדר אותן כך שירכיבו ריבוע; בהינתן קוביה הונגרית במצב x, מטרת השחקן היא למצוא סדרת מהלכים y שתחזיר אותה למצב מסודר, וכן הלאה. כל המשחקים הללו מאופיינים בכך שהם משחקים לשחקן **יחיד** ועם ידע מלא (אין אלמנט של אקראיות, מרגע שהשחקן קיבל את הקלט).

הבעיה ה־NP־שלמה המייצגת ביותר היא SAT. אפשר לחשוב על SAT בתור בעיה של בדיקת מחרוזות בייצוג בינארי: השמה היא מילה $w\in\{0,1\}^{n}$ ופסוק ה־SAT φ בודק האם המילה w עונה על קריטריון מסוים ש־φ "מקודדת". כפי שראינו, לכל המשחקים שלעיל אנו יכולים לבנות נוסחה φ שתקודד את המשחק עצמו, ותבדוק האם w היא פתרון של המשחק.

את הרעיונות הללו ניתן להכליל למשחקים בשני שחקנים; המחלקה המתקבלת היא PSPACE והשפה המקבילה ל־SAT במחלקה זו היא TQBF.

על TQBF ניתן לחשוב כעל השפה של פסוקים לוגיים **עם כמתים**. כמתים הם סימנים המוצמדים לפסוק ומוסיפים לו משמעות של **לכל** או של **קיים**. נתאר זאת פורמלית.

בהקשר שלנו "פסוק" יכלול משתנים, קשרים, **קבועים** (0 ו־1) וכמתים, כשכמתים מתווספים בצורה הבאה:

הגדרה 15.6 אם φ הוא פסוק, אז $\forall x\varphi$ (קרי "לכל x φ") הוא פסוק ו־$\exists x\varphi$ (קרי "קיים x כך ש־φ") הוא פסוק. הסימנים $\forall,\exists$ נקראים **כמתים** ונשתמש בסימון Q כדי לציין כמת מבין $\forall,\exists$ באופן כללי.

אומרים שבפסוק $Qx\varphi$, כל מופע של x בתוך φ **נופל תחת הכמת** Q.

משתנה שנופל תחת כמת כלשהו נקרא **משתנה קשור** ומשתנה שלא נופל תחת אף כמת נקרא **משתנה חופשי**.

אם בפסוק אין משתנים חופשיים, ערך האמת שלו נקבע באופן חד משמעי, בצורה הבאה:

הגדרה 16.6 ערך האמת של פסוק שכולל רק קבועים וקשרים נקבע בהתאם לטבלאות האמת של הקשרים.

נסמן ב־$\varphi\left(x\right)$ פסוק עם משתנה חופשי x ונסמן ב־$\varphi\left(b\right)$ את מה שמתקבל מהפסוק כשמחליפים כל מופע של x בקבוע b ($b\in\{0,1\}$), אז:

- ערך האמת של $\forall x\varphi(x)$ הוא T אם ורק אם ערך האמת של $\varphi(0)$ וגם של $\varphi(1)$ הוא T.
- ערך האמת של $\exists x\varphi(x)$ הוא T אם ורק אם ערך האמת של $\varphi(0)$ או של $\varphi(1)$ הוא T.

נאמר שפסוק הוא QBF (Quantified Boolean Formula) אם הוא מהצורה $Qx_1Qx_2\ldots Qx_n\varphi(x_1,\ldots,x_n)$ כך ש־φ הוא פסוק שהמשתנים היחידים שלו הם $x_1,\ldots,x_n$. לפסוק כזה יש ערך אמת מוגדר: או T או F.

הגדרה 17.6 השפה TQBF כוללת את כל פסוקי ה־QBF שערך האמת שלהם הוא T.

נשים לב שאפשר לחשוב על SAT בתור מעין מקרה פרטי של TQBF שבו כל ה־Q־ים הם $\exists$, ואז השאלה האם קיימת לפסוק השמה מספקת היא למעשה השאלה האם הפסוק עם הכמתים הוא T.

משפט 18.6 $\mathrm{TQBF}\in\mathrm{PSPACE}$.

הוכחה: נראה אלגוריתם רקורסיבי שבכל קריאה רקורסיבית מוריד את אחד מהכמתים של הנוסחה. תנאי העצירה הוא נוסחה φ ללא כמתים ־ בנוסחה כזו אין משתנים כלל (כי ב־QBF אין משתנים חופשיים) אלא רק קבועים, וחישוב ערך האמת של הפסוק הוא פולינומי בגודל הפסוק.

בהינתן פסוק מהצורה $Qx\varphi(x)$ (כאשר φ עצמו עשוי להכיל כמתים נוספים) האלגוריתם יפעל כך:

- אם $Q=\exists$, האלגוריתם ירוץ רקורסיבית על $\varphi(0)$. אם התקבל שערך האמת של $\varphi(0)$ הוא T, האלגוריתם יחזיר T. אחרת, הוא ירוץ רקורסיבית על $\varphi(1)$ ויחזיר את הפלט.
- אם $Q=\forall$, האלגוריתם ירוץ רקורסיבית על $\varphi(0)$. אם התקבל שערך האמת של $\varphi(0)$ הוא F, האלגוריתם יחזיר F. אחרת, הוא ירוץ רקורסיבית על $\varphi(1)$ ויחזיר את הפלט.

עומק הרקורסיה בהרצה של האלגוריתם על פסוק עם n כמתים הוא n, וכל רמה של הרקורסיה דורשת ביט בודד של זיכרון למעט שלב תנאי העצירה שדורש זיכרון פולינומי. מכאן שהאלגוריתם כולו דורש רק זיכרון פולינומי. ■

הסיבה שבגללה TQBF מעניינת אותנו היא ההקבלה בינה ובין SAT. כזכור, SAT הייתה שפה NP־שלמה. בדומה, TQBF תהיה שפה PSPACE־שלמה.

הגדרה 19.6 שפה $L\in\mathrm{PSPACE}$ היא PSPACE־שלמה אם לכל $L'\in\mathrm{PSPACE}$ מתקיים $L'\le_p L$.

הרדוקציה שבהגדרה היא רדוקציה פולינומית רגילה, כלומר כזו שניתנת לחישוב **בזמן** פולינומי, לא במקום פולינומי. הסיבה לכך היא שהרדוקציה מוגדרת מתוך מחשבה לא על המחלקה ה"גדולה" (PSPACE) אלא על המחלקה ה"קטנה" (P) שמעניינת אותו בהקשר של השאלה האם $\mathrm{P}=\mathrm{PSPACE}$:

משפט 20.6 אם L היא שפה PSPACE שלמה אז $\mathrm{P}=\mathrm{PSPACE}$ אם ורק אם $L\in\mathrm{P}$.

הוכחה: אם $L\notin\mathrm{P}$ אז בפרט $\mathrm{P}\ne\mathrm{PSPACE}$. אם $L\in\mathrm{P}$ אז לכל $L'\in\mathrm{PSPACE}$ מתקיים $L'\le_p L$ וממשפט הרדוקציה נקבל $L'\in\mathrm{P}$. ■

נותר להסביר מדוע TQBF היא PSPACE־שלמה. נציג את רעיון ההוכחה אך לא ניכנס באופן מלא לפרטים.

משפט 21.6 TQBF היא PSPACE־שלמה.

הוכחה: תהא $L\in\mathrm{PSPACE}$ עם מכונה בעלת זיכרון פולינומי M שמכריעה אותה. בהינתן x, נרצה לבנות פסוק TQBF ψ שערך האמת שלו הוא T אם ורק אם $x\in L$.

בהוכחת משפט קוק, נעזרנו בקידוד של הריצה של M על x בתור טבלת אמת. ייצרנו משתנים שמתארים כל צעד בריצה, תוך הסתמכות על כך שזמן הריצה הוא פולינומי. כאן המכונה M בהחלט עשויה לפעול בזמן ריצה אקספוננציאלי, ולכן אין שום דרך לייצר משתנים אינדיבידואליים עבור כל צעדי הריצה כמו במשפט קוק מבלי לחרוג מגבולות הזמן הפולינומיים של הרדוקציה. לכן ננקוט ב"תעלול" שנעזר בכמתים שיכולים להופיע בנוסחה.

נשים לב לכך שכדי לקודד **קונפיגורציה** של M בריצתה על x די בזיכרון פולינומי, וניתן לכתוב פסוק פולינומי חסר כמתים שבודק עבור זוג קונפיגורציות C,C' האם קיים מעבר מ־C אל C', בשיטה דומה לזו שבה בוצעה בדיקה זו בהוכחת משפט קוק. מה שנעשה הוא לבנות סדרת פסוקים $\psi_0,\psi_1,\psi_2,\ldots$ כך שבהינתן הפסוק $\psi_k(\overline{x},\overline{y})$ עם קבוצות המשתנים החופשיים $\overline{x},\overline{y}$, אם נציב במשתנים אלו את הערכים שמקודדים זוג קונפיגורציות C,C', נקבל פסוק QBF $\psi_k(C,C')$ שהוא T אם

ורק אם קיים מסלול מאורך 2^k בגרף הקונפיגורציות של M על x שמוביל מ־C אל C'. הפסוק ψ_0 הוא פשוט הפסוק שבודק האם קיים מעבר ישיר מ־C אל C', ואילו עבור ערך גדול מספיק של k, ψ_k יהיה בדיוק ψ שאנו רוצים לבנות.

לצורך כך, ראשית נוודא שבגרף הקונפיגורציות יש קונפיגורציה מקבלת יחידה שנסמן C_{accept} - למשל, אחרי שהמכונה הבינה שעליה לקבל היא מרוקנת את כל הסרט, מעבירה את הראש הקורא לתחילת הסרט ועוברת למצב המקבל. כמו כן למכונה יש קונפיגורציה התחלתית יחידה C_{start}. לבסוף, בגרף הקונפיגורציות נוסיף מעבר מ־C_{accept} לעצמה, כך שאם קיים מסלול באורך **קטן או שווה** ל־2^k, יהיה גם מסלול שאורכו שווה בדיוק ל־2^k.

כעת, יהא m החסם על הזיכרון בריצת M על x, כלומר גודל כל קונפיגורציה הוא m, ולכן על פי ספירת קונפיגורציות האורך המקסימלי של מסלול מקבל יכול להיות 2^m, כלומר $\psi_m\left(C_{start},C_{accept}\right)$.

נותר להסביר כיצד נבנה באופן אינדוקטיבי את $\psi_0,\psi_1,\ldots$. את הבסיס כבר יש לנו, אז נניח שבנינו את ψ_k ונבנה את ψ_{k+1}. ראשית נציג את הבניה האינטואיטיבית **שלא תעבוד**: קיים מסלול מאורך 2^{k+1} מ־C אל C' אם ורק אם קיים **מצב ביניים** C'' שקיים אליו מסלול מאורך 2^k מ־C, וקיים מסלול ממנו אל C' שגם הוא מאורך 2^k, כלומר היינו רוצים להגדיר $\psi_{k+1}\left(C,C'\right)=\exists C''\left(\psi_k\left(C,C''\right)\wedge\psi_k\left(C'',C'\right)\right)$

מדוע בניה זו אינה טובה? הפסוק אכן מקיים את המבוקש ממנו, אבל הוא **גדול מדי**. על מנת לבנות את ψ_{k+1} אנו משתמשים בשני עותקים של ψ_k, מה שמוביל לפסוק בגודל אקספוננציאלי. כדי שהבניה תעבוד עלינו להשתמש בצורה יצירתית יותר בכמתים שעומדים לרשותנו כדי לחסוך לעצמנו שימוש אחד של ψ_k. נשים לב שעד כה לא השתמשנו כלל בכמת $\forall$.

מה שנעשה הוא לבנות פסוק שאומר שקיים C'' כך שלכל C_1,C_2, **אם** $C_1=C$ וגם $C_2=C''$ אז קיים ביניהם מסלול מאורך 2^k, **ואם** $C_1=C''$ וגם $C_2=C'$ אז קיים ביניהם מסלול מאורך 2^k:

$$\psi_{k+1}\left(C,C'\right)=\exists C''\forall C_1\forall C_2\left[\left(C_1=C\wedge C_2=C''\right)\vee\left(C_1=C''\wedge C_2=C'\right)\right]\to\psi_k\left(C_1,C_2\right)$$

פסוק זה מוסיף מספר קבוע של משתנים כדי לקודד את C'',C_1,C_2 ולכן אורכו הכולל הוא $|\psi_{k+1}|\le O\left(m\right)+|\psi_k|$, כך שנקבל ש־$|\psi_m|\le O\left(m^2\right)$ - ריבועי, ובהחלט ניתן לייצור במסגרת רדוקציה פולינומית, כמבוקש. הקושי היחיד הוא בכך שפסוק זה אינו בצורת QBF כי הכמתים אינם כולם "בחוץ". למרבה המזל קיים אלגוריתם שמעביר פסוק כלשהו עם כמתים לצורה שבה כל הכמתים בחוץ (צורת Prenex) שפועל בזמן פולינומי, והוא יסיים לנו את הבניה. ■