/epstopdf-sys.cfg /epstopdf.cfg

סיכום AI

25 ביולי 2021

1 שבוע 1:

1.1 הרצאה 1:

- **היוריסטיקה** $h(n)$: זוהי שיטת חיפוש המתקדמת לפי שיקולים מוגדרים מראש. היא לא בהכרח מתקדמת ליעד ספציפי אך חותרת הגיע אליו באופן כללי. כלומר - תמיד נתקדם לכיוון הכללי של היעד.
  עבור כל צומת - נצמיד לה מספר המעריך מה הסיכוי להגיע אל היעד דרך הצומת הנוכחית.
  זאת פונקציה שמקבלת נקודה, ומחזירה לנו מספר המציין כמה אנחנו רחוקים או קרובים למטרה.

- $\boldsymbol{A^* \ search}$: אלגוריתם חיפוש בעזרת היוריסטיקה, האלגוריתם עובר על צמתי גרף ממושקל ומוצא את המסלול הקצר ביותר מהמקור ליעד.
  פונקציית האלגוריתם מוגדרת באופן הבא: $f(n) = h(n) + g(n)$. עבור :
  $f(n)$: פונקציית העלות. $h(n)$: זוהי פונקציית היוריסטיקה אדמיסבילית - הערכה למרחק בית הצומת הנוכחית לצומת היעד. $g(n)$: סכום המחירים של כל המעברים שעשינו מצומת ההתחלה עד הצומת הנוכחית.
  אלגוריתם זה עובד בעזרת שימוש בערימות - $queue$

1.2 תרגול 1:

- **הנחות שאנו מניחים בכדי לפתור בעיות ע"י בינה מלאכותית:**
  1: אנו מייחסים לבינה המלאכותית שלנו כאילו הוא השחקן היליד בתמונה.
  2: הכל נשאר סטטי קבוע ולא משתנה.
  3: אנו יודעים מה תהיה התוצאה בהתאם לפעולות שנבצע, אם נזוז משבצת ימינה נגיע לימין.
  4: הכל ידוע מראש ואין הפתעות.
  5: מרחב המצבים שלנו הינו סופי.

- **הגדרות בחיפוש:**
  1: העולם שבו אנו משחקים.
  2: הפעולות אותן אנו אמורים לבצע כדי להתקדם.

3: נקודת ההתחלה והסיום.

4: הפתרון אותו אנו צריכים לבצע - איזה פעולות עשינו בכדי לפתור את המשחק או המסלול.

- בכדי לפתור בעיה מורכבת ננסה לסנן את המידע ולהשאיר רק את המידע הנחוץ. לדוגמא - באילו משבצות נוכל לעבור ובאילו לא, נסמן משבצות מעבר בלבן ומשבצות אסורות (קיר או בור) באדום.

- אנו נרצה לכתוב פונקציה שאומרת לנו מצבים מסויימים אילו פעולות מסויימות עלינו לבצע בכדי לפתור את הבעיה.

- בנוסף אנו נצטרך לכתוב את מחיר הפתרון - מה עלות הפתרון להגיע מנקודת ההתחלה לנקודת היעד.

- **בעית חיפוש פורמלית:** $\langle S, s_0, G, A, F, C \rangle$

  S: כל המצבים.

  s_0: מצב התחלתי.

  G: קבוצת של המטרות אליהן נרצה להגיע (חניות פנויות לדוגמה).

  A: קבוצת הפעולות אותן אנו יכולים לבצע.

  $F : S \times A \Rightarrow S$: פונקצית המצבים - עבור איזו פעולה נצטרך לבצע עבור כל מצב

  $C : S \times A \Rightarrow \mathbb{R}^+$: פונקצית העלויות של ביצוע כל פעולה.

באופן פורמלי, **פתרון** הינו הזוג הבא: $\left\langle \{s_i\}_{i=0}^{n}, \{a_i\}_{i=0}^{n-1} \right\rangle$

המחיר הכללי: באופן פורמלי הוא סכום כל המעברים - $\sum_{i=0}^{n-1} C\left(s_i, a_i\right)$.

בכדי שפתרון יהיה חוקי הוא צריך להתחיל בנקודת ההתחלה ולסיים ביעד כלשהו, בנוסף - כל מעבר בדרך צריך להיות חוקי גם כן.

- **כיצד נממש אלגוריתם חיפוש:**

  1: נגדיר תחילה מהו העולם שבו אנו משחקים.

  2: נבחר את מבני נתונים שבו נשמור את המידע - מאיפה הגענו, מה המצב העכשווי וכו...

  3: נגדיר פונקציה $L : V \Rightarrow S$ האומרת לנו באיזה מצב אנו נמצאים.

עץ חיפוש: בד"כ את החיפוש אנו נממש בעזרת עץ מכוון. שבו כל צומת n מייצגת לנו מצב ייחודי, כאשר השורש מיצג לנו את המצב ההתחלתי, והבנים מייצגים לנו את המצבים אליהם ניתן להגיע מהאב, והצלעות מייצגות מעבר בין מצבים.

ניתן לייצר ילד לקודקוד כאשר יש לנו מצב חדש שלא מופיע בעץ. אותו נגדיר כ n', אך יש לשים לב לא לייצר $state$ חדש סתם, אם קיימת אופציה לשים מצביע או לבצע $shallow\ copy$ במקום $deep\ copy$.

- ישנם מצבים שבהם נעדיף ממש את החיפוש בעזרת גרף ולא בעזרת עץ.

- **כיצד נכריע האם אלגוריתם החיפוש טוב או לא:**

  1: שלמות - אם יש פיתרון, האם האלגוריתם ימצא אותו.

  2: נאותות - במקרה שאין פתרון, האם האלגוריתם יסיים ויכריע שאין פתרון.

3: אופטימליות - האם האלגוריתם ימצא פתרון אופטימלי (לרוב נעדיף למצוא פתרון בזמן סביר, מאשר פתרון

אופטימלי).

4: סיבוכיות - מהי סיבוכיות הזמן כדי להגיע לפתרון, ומספר הצמתים שייצרנו. בנוסף סיבוכיות המקום חשובה גם כן.

- **פרמטרים חשובים בסיבוכיות:**

  b: קצב ההתרחבות של מצבים חדשים, עבור כל פעם שנרחיב את העץ

  d: עומק הפתרון הזול, האופטימלי והקרוב

  m: העומק המקסימלי של המצבים שיש לנו (יכול להיות אינסוף).

- $\boldsymbol{Defth - Limited - Search}$: זהו אלגוריתם שעובד כמו DFS אך הוא מחפש עד העומק הקצר ביותר. (ישנם אופציות שהפתרון נמצא בגרף אך האלגוריתם לא יגיע אליו.) היתרון של האלגוריתם הזה זה שהוא רץ מהר מאד.

  ניתן להפעיל אותו כמה פעמים אחרי שנמחק את כל הענפים שביקרנו בהם, ונגדיל בכל פעם את החיפוש באחד, וכך נמצא את הפתרון. כך נקבל אלגוריתם שלם, נאות ואופטימלי.

2 שבוע 2:

2.1 הרצאה 2:

- בבינה מלאכותית הרבה בעיות יכולות להיפתר בעזרת חיפוש שאותו נבצע בעזרת היוריסטיקה.

- A^*: הוא אלגוריתם חיפוש מושלם, ואופטימלי.

  משפט: באלגוריתם הנל יתקיים תמיד:

- SMA^*: אלגוריתם שבנוי על A^* אך בשונה ממנו הוא משתמש בכל הזיכרון הזמין.

  האלגריתם פועל כך:

  1: הוא מוסיף את כל הקודקודים עד שנגמר הזיכרון.

  2: מוציא את כל הקודקודים הגרועים.

  3:

- $\boldsymbol{locsl\ search\ algorithems}$: אלגוריתמי חיפוש שמחזירים רק את נקודת היעד בלי התחשבות במסלול. משום שאנו מחפשים את היעד ולא מעניין אותנו המסלול.

  כיצד האלגוריתם פועל: הוא משתמש בנקודה הראשונה ובודק בכלפעם האם ניתן לשפר אוה על ידי הזזה למשבצת הקרובה.

  יתרונות: אנו משתמשים בפחות זיכרון, ניתן למצוא פתרון טוב גם בלוח ענק.

- **בעית המלכות:** אנו מקבלים לוח שחמט עם כמה מלכות. המטרה היא להזיז אותן בלוח כך שאף מלכה לא תאיים על האחרת.

  קיימת $fitness\ function$ אשר אומרת לנו בכל שלב כמה זוגות של מלכות מאיימות אחת על השניה. ניתן לפתור בעיה זו עם מספר אלגוריתמים שונים.

2.2 תרגול 2:

- **בכדי לממש חיפוש אנו נשתמש בשתי פונקציות:** $g(v)$: סכום המחירים של כל המעברים שעשינו מצומת ההתחלה עלד הצומת הנוכחית. $h(v)$: עלות מהצומת המוכחית עד ההגעה למטרה. $f(v) = g(v) + h(v)$: זהו מסלול שלם מההתחלה עד לנקודת היעד.
  למעשה: $g(v) = \sum_{i=0}^{n-1} C\left(s_i, a_i\right)$.

- $\boldsymbol{h_{SLD}(V)}$: זוהי היוריסטיקה המתאימה לניווט על מפה. היא בודקת את המרחק מהצומת הנוכחית x, ליעד בעזרת קו מרחק אווירי.

- $\boldsymbol{best\ first\ search}$: רעיון האלגוריתם הוא להשתמש בפונקצית הערכה - היוריסטיקה, ולהגיע ליעד בדרך הקצרה ביותר.
  גרסה חמדנית: קיימת גרסה חמדנית לאלגוריתם הנ"ל. האלגוריתם החמדן **לא** מושלם, **לא** נאות, **סיבוכיות זמן ומקום:** $O(b^m)$, בנוסף הוא **לא** אופטימלי.

- $\boldsymbol{A^*\ search}$: זהו האלגוריתם העיקרי שבו נשתמש. והוא עובד בצורה הבאה - ברגע שהוא יראה שקודקוד מסויים לא מקדם אותנו ליעד - נזנח אותו ונעבור למסלול אחר.
  כיצד יעבוד האלגוריתם: נוסיף את כל הצמתים החדשים לתור ואף פעם לא נעשה $reset$. ברגע שקודקוד מסויים מקדם אותנו ליעד - נלך דרכו. אם נראה שלבסוף הוא לא מקדם אותנו - נעזוב אותו ונחזור לקודקודים אחרים שכן סביר שיקדמו אותנו ליעד.

- **הגדרה - פונקציה אדמיסבילית:** פונקציה אופטימית. כלומר פונקציה שאף פעם לא תתן לנו מחיר גדול מהמחיר האמיתי של המסלול . כלומר תמיד מתקיים: $h(v_0) \leq \sum_i^n c(e_i)$

- **הגדרה - פונקציה** $\boldsymbol{consistent}$ **(הגיונית):** כמו אי שוויון המשולש - המרחק בין v ל v' קטן יותר מאשר $h(v)+h(v')$. ובאופן פורמלי :

$$\forall e = (v, v') \in E \Rightarrow h(v) \leq c(e) + h(v')\ and\ \forall v \in G \Rightarrow h(v) = 0$$

- **הגדרה -**$\boldsymbol{dominates}$**:** נאמר ש h_2 יותר טובה מ h_1 אם היא קיימת לכל v: $h_2 \geq h_1$.

- **טענה:** אם פונקציה היא קונסיסטנטית אזי היא בהכרח גם אדמיסבילית (הכיוון השני לא בהכרח נכון).

- **תכונות של** $\boldsymbol{A^*}$**:** האלגוריתם **מושלם**, **לא** נאות, **סיבוכיות זמן:** אקספוננציאלי ביחס הטעות של h כפול אורך הפתרון. **סיבוכיות מקום:** $O(b^m)$. הוא **אופטימלי** בחיפוש בעץ (עבור גרף - נצטרך שהפונקציה תהיה $consistency$).

- **שיטות כיצד לפתח היוריסטיקה:**
  1 $\boldsymbol{relaxation}$**:** אנו לוקחים את הבעיה המקורית ומתעלמים מחלק מההגבלות על הבעיה. (לדוגמא אם מותר לנו לזוז רק דרך משבצות מסויימות - נתעלם מזה ונתייחס רק לחלק מההגבלות).
  $\boldsymbol{abstraction}$2**:** נסתכל על תת בעיה ונפתור אותה, תוך התעלמות מהבעיה הכללית.

- **שילוב היוריסטיקות:**

1 ***relaxation/abstraction:*** נוריד חלקים מהבעיה המקורית ונפתור את תת הבעיה **באופן מדוייק**, ונספור כמה צעדים לקח לנו כדי להגיע לפתרון. לאחר מכן נשתמש במספר הצעדים הנ"ל בכדי לפתור את הבעיה הכללית.

2 ***wighting:*** אם יש לנו כמה היוריסטיקות אדמיסביליות אנו יכולים לקחת ממוצע משוקלל של המשקל של כולן, כך שסכומן הוא 1. ואז נקבל היוריסטיקה חדשה - **אדמיסבילית**.

3 ***composition:*** שימוש בהיוריסטיקה המקסימלית - אנו חייבים שכל אחת מהן תהיה דומיניייטד. (אך לא בהכרח שנקבל היוריסטיקה אדמיסבילית - ואז הפתרון לא יהיה אופטימלי).

- **חיפוש מקומי:** נועד לפתור בעיות אופטימיזציה קשות - כאשר לא אכפת לנו מהמסלול, ולא יודעים מראש מה המטרה, והפתרון אינו מסלול אלא הוא ה *state* הכי טוב שאנו יכולים למצוא. בנוסף יכול להיות שאין אפילו נקודת התחלה. (דוגמא טובה - חיפוש מקום לקליטת רדיו בג'ונגל *hill climing*).

  כיצד נממש: בכל נקודה שאנו נמצאים נפתח קודקודים, נבחר באחד ונזרוק את הישנים.

- **אלגוריתמים לחיפוש מקומי:**

:1 ***gardient descent:*** נלך לכיוון הכללי אליו אנו רוצים להגיע. (בדוגמא למעלה נחפש את הנקודה הגבוהה ביותר). **הבעיות:** בטעות אנו יכולים להגיע לאיזור שטוח ואז לא נדע לאן להתקדם, או למקסימום מקומי שאחריו יש עמק. (בפתרון בעיה *hill climing* אנו מנסים להימנע מבעיית *local minimum\max*)

כיצד נממש: נבחר נקודה אקראית. ונבדוק בעזרת לולאה אם האיבר הבא יותר טוב מהמצב הנוכחי : אם כן - נחליף ונתקדם. אם לא - נלך לכיוון אחר. ובסוף נמחק את כל ה *frange* הישן.

:2 ***local beam search :*** במקום לחפש מנקודה אחת, נחפש מכמה נקודות במקביל. אמנם נשתמש ביותר זיכרון, אך נגיע לפתרון מהר יותר.

3 שבוע 3:

3.1 הרצאה 3:

- ***simulated annealing:*** בדומה לדוגמה שהבאנו בתרגול 2. כאשר אנו מחפשים נקודה גבוהה בג'ונגל בכדי לחפש מקוסלקליטת רדיו, האלגוריתם יעבוד באופן הבא: אנו נתחיל לעלות ונרשה לעצמנו לרדת בחזרה בתקווה למצוא מקום גבוה יותר בעתיד. אך ככל שהזמן מתארך אנו מרשים לעצמנו לרדת פחות ופחות בכדי שנישאר בגובה. למעשה אנו מתחילים עם הפרש גדול ומאפשרים מרווח ירידה שעם הזמן מצטמצם.

  אם הפרשי הגובה אנו נמדוד בעזרת הטמפרטורה, שיורדת כאשר אנו עולים בגובה.

- ***traveling saleaman problam:*** יש לנו מפה עם מספר ערים, ואנו צריכים לעבור בכולן כך שנבחר את הדרך הקצרה ביותר. בעיה זו ניתן לפתור בעזרת האלגוריתם מלמעלה - *simulated annealing*. אנו נבחר נקודות רנדומלית ואחכ נחליף ביניהן, עם מרווח טעות שפוחת בכל פעם.

- ***Genetic algorithems:*** בדומה ל *simulated annealing* הוא מחפש את הנקודה הבאה ונמנע מנקודות מינימום\מקסימום מקומיות. אך הוא מאפשר לדלג על נקודות . אנו לוקחים קטעים ולא נקודות - אנו בוחרים את

הצעד הבא באופן אחר. אנו מסתכלים על שני ההורים של הקודקוד, משלבים אותם יחד (את המצבים הטובים ביותר מכל הורה) ורואים מה התוצאה, ולפי זה יודעים היכן המיקום של הנקודה הבאה, ומהו הצעד הבא אותו אנו צריכים לבצע.

בשיטה זו ניתן לפתור את בעית המלכות: אנו משלבים בין שורות ועמודות של שני מצבים (ההורים), ויודעים מי יהיה ה "ילד", פעם נקח שורה מלוח ימין ופעם עמודה מלוח שמאל ונשלבם יחד. ייצוג הבעיה הוא בעזרת מספרים הממוקמים על הלוח ושילוב חצי בעיה מכל לוח. את ההחלטה כיצד לשלב את המלכות נבצע בעזרת $fitness\ function$ ונבחר את המהלכים עם ערך ההחזרה של הפונקציה הגבוה ביותר.

כאשר אנו מגיעים למינימום מקומי האלגוריתם ימצא את פתרון הבעיה בעזרת "מוטציות". בכל פעם יהיו מספר מהלכים שמכוונים למקום אחר שלא בהכרח מכוון אותנו לפתרון, אך אם צעדים אלו אכן יקרבו אותנו לפתרון אנו בסוף נבחר בהם. (בעיה הנחש והעכבר).

- **בעיית צביעת מפת אוסטרליה:** יש לנו שלשה צבעים ושישה מחוזות. ואנו רוצים לצבוע את מפת אוסטרליה כך שלא יהיו שני מחוזות צמודים הצבועים באותו הצבע.

- **בעיות חיפוש ספציפיות:**

1 בעיית חיפוש סטנדרטית: יש לנו $state$: שהיא קופסה שחורה המייצגת את העולם שלנו. $succesor\ func$: בכדי להגיע לילדים. $heuristic\ func$: שתאמר לנו באיזו דרך עדיף לנו להתקדם. $goal\ state$: נקודת הסיום.

2 CSP- בעיות אילוצים: $state$: מיוצג בעזרת משתנים X_i עם ערכים D_i (ערכים שכל משתנה x_i יכול לקבל). $goal\ state$: סט של אילוצים המתקבלים מהמשתנים - סט של אילו מהלכים הם חוקיים.

3 $backtreacking\ search$: אנו מקבלים את מפת אוסטרילה ורוצים לצבוע אותה, אך במקום לפתוח עץ עם כל המצבים בידיעה שאנו מוסיפים מראש מצב שאינו חוקי. אנו נוסיף את המצבים רק אם הם חוקיים.

שיפור אלגוריתם בק־טרקינג: את האלגוריתם הנ"ל ניתן לשפר בעזרת היוריסטיקות, הנה כמה מהן:

MRV: היוריסטיקה שאומרת לבחור תחילה מחוז אחד שאותו נצבע, ואחכ נצמצם את האפשרויות על ידי צביעה של שאר המחוזות לפי החוקיות. כלומר - בכל פעם נצמצם אט אט את האפשרויות שלנו.

$degree\ heuristic$: היורסטיקה זו בוחרת את ראשון את המחוז עם **הכי הרבה** מגבלות צביעה, וצובעת אותו ראשון. ואחכ מתקדמת לפי מספר המגבלות בסדר יורד.

$least\ constraining\ value$: היורסטיקה זו דוגלת בבחירת הערך עם **הכי פחות** מגבלות צביעה, ואחכ נתקדם בסדר עולה.

- **הימנעות מענפים אשר יביאו אותנו ל $dead-end$:**

$Forward\ checking$: הרעיון הוא להפחית את האופציות הרעות בכל שלב. וכך כאשר יש לנו ענף שלא מוביל אותנו לשום מקום - לא נוסיף אותו כלל, אלא נוסיף בכל פעם רק קודקודים רלוונטים. בכל שלב נבדוק **רק** את השכן, ולא יותר.

$Constraint\ propagation$: בניגוד לשיטה הקודמת, בשיטה זו אנו **כן** מסתכלים על השכן של השכן. כך למעשה אנו מגלים $dead-end$ מוקדם יותר מאשר בשיטה של $Forward\ checking$.

$Arc\ concisency$: זו הרחבה שאומרת במקום לבדוק רק את מי שמסביב לערך ולמחוק מהדומיין שלו משתנים לא מתאימים, אנו נבדוק את ההמשך ונמחק כבר הלאה את כל האילוצים שלא מתאימים.

באופן פורמלי: עבור שני ערכים X,Y נאמר שהם קונסיסטנטים אמ"מ לכל $x \in X$ קיים $y \in Y$ הגיוני.

היא עובדת רק עבור בעיות סיפוק אילוצים בינארים.

סיבוכיות: $O(m \cdot d^3)$. כלומר - הרחבת הקודקודים קטנה, אך זמן הריצה עדיין נשאר גדול.

- $Games\ search$: במשחק, לכל שחקן יש מטרה למקסם את רווחיו ולהפחית את רווחי היריב.

$minmax$: בשיטה זו - בכל מהלך שלנו אנו נבחר את המהלך שיחזיר לנו את המקסימום האפשרי.

$\alpha - \beta$: שיטה זו נועדה לצמצם את מספר תתי העצים שאנו פותחים בכל שלב, והיא תעבוד באופן הבא: אם הקודקוד הוא קודקוד של שחקן ה'מקס' - זאת אומרת, זהו תורו של 'מקס' לשחק - השחקן לא יבחר תתי־עצים בעלי תוצאה נמוכה יותר מזו שהושגה בתת־עץ קודם. הרציונל לכך הוא פשוט. זהו תורו של 'מקס', 'מקס' תמיד בוחר למקסם. בשל כך, ברור כי אם 'מקס' כבר יודע על מהלך בעל ניקוד מסוים, אם באחד התתי עצים שלו שחקן ה'מין' הצליח להשיג ניקוד נמוך יותר על ידי אחד התתי עצים שלו עצמו, שחקן ה'מין' לא צריך לבחון את שאר תתי העצים שלו משום שהוא לא יבחר ניקוד גבוה יותר ממה שהשיג כעת ולכן שחקן ה'מקס' (שהוא אב קדמון שלו) לא יבחר בניקוד המגיע ממנו כי יש לו כבר ניקוד גבוה יותר.

- $MCTS-monte\ carlo\ tree\ search$: יש לנו $valuetion\ func$ שנותנת לנו ערך עבור כל מהלך (לדוגמה במשחק שחמט - עדיף לאבד שחקן רגיל בכדי שהיריב יאבד מלכה). וכך אנו מתקדמים לפי המהלך המשתלם ביותר. חילה נשחק את המשחק פעם אחת באופן רנדומלי, ואחכ נבדוק אילו מהלכים הובילו אותנו לנצחון ואילו להפסד, וננקד את המהלכים בהתאם.

3.2 תרגול 3:

- **אלגוריתם גנטי:** זהו סוג של אלגוריתם חיפוש מקומי שבו אנו מחפשים את נקודת הסיום הטובה ביותר ללא התחשבות במסלול.

מתי נשתמש באלגוריתם גנטי: כמעט כל בעיית אופטימיזציה ניתן לפתור בעזרת אלגוריתם גנטי, אך זו דרך גרועה. לכן נשתמש בהם כשיש לנו בעיה שאין דרך אחרת לפתור אותה.

מוטיבציה: אלגוריתמים אלו הם הרחבה של $simulated\ annealing$ שבו יש לנו פתרון אחד ואנו מוכנים לעשות מהלך גרוע בכדי להגיע לפתרון טוב יותר בהמשך. באלגוריתם זה, אנו נקח חלקים מכל פתרון ונחבר יחד את אלו שעובדים הכי טוב.

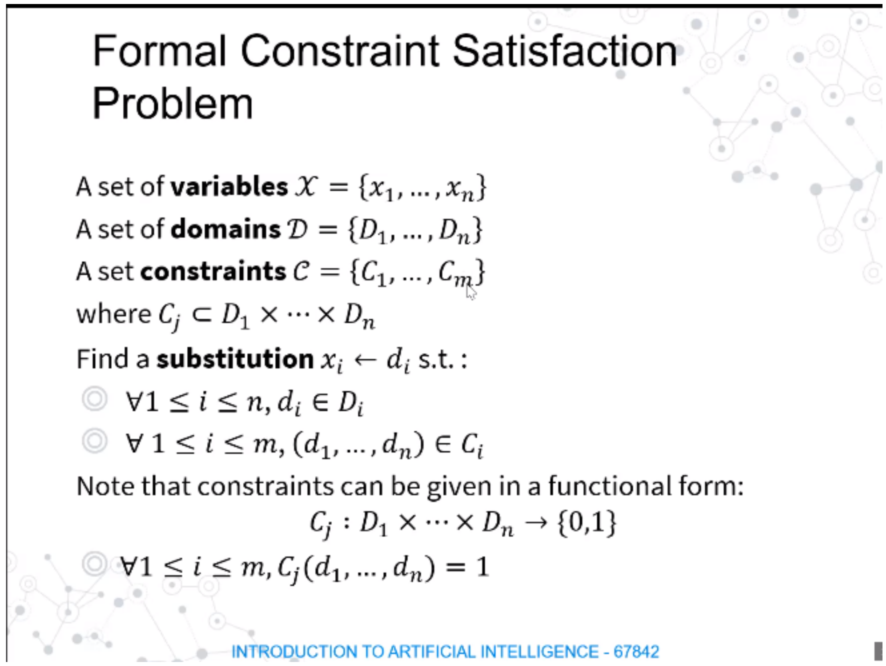
מימוש: אנו מקבלים כקלט וקטורים שמייצגים את המצבים. לפי $fitness\ func$ אנו נחליט אלו חלקים מהווקטור נקח מכל הורה, ונחבר. לבסוף נוכל לתקן בעזרת מוטציות שעושות שינויים קטנים.

הערה: אנו לא נקח בכל שלב את ההכי טובים, כדי שלא נתקע במקסימום מקומי. לכן נקח גם פרטים קצת פחות טובים בכדי שה"אוכלוסיה" תהיה מגוונת.

נבצע זאת בעזרת רולטה, וככל שהפרט טוב יותר נתן לו חלק גדול יותר במעגל. וכשנסובב את הרולטה הפרטים עם הגנים הטובים יותר יהיו בעלי הסתברות גבוהה יותר להיבחר.

- CSP- **בעיות אילוצים:** סודוקו, צביעת גרף בצבעים שונים, סידור מטלות וכו'...

פורמלית:



מימוש האלגוריתם: נגדיר גרף אילוצים שבו הצמתים הם משתנים, והצלעות הן אילוצים בין קודקודים (בין כל זוג קודקודים עם אילוץ תהיה צלע).

אנו נתחיל במשתנה מסויים לפי היוריסטיקה שנבחר ונעתיק את הדומיין של המשתנה, כל עוד לא נגמרו לנו משתנים נבחר ערך מהדומיין שנבחר בהתאם לאילוצים שכבר הצבנו. אם לא הצלחנו להציב - נחזור אחורה. אם לא נשארו לנו ערכים, נעבור למשתנה הבא. אחרי שנסיים עם ערך נמחק אותו מהדומיין.

select value - פונקציה שבוחרת איזה ערך לשים בהתחשב באילוצים שכבר הצבנו, אם יש בעיה - נלך לערך אחר. אם לא - נציב אותו.

טענה: כל בעיית סיפוק אילוצים כללית ניתן להפוך לבעיית אילוצים בינארית.

- ***Thtashing***: אלגוריתם בק־טרקינג שחוזר בכל שלב רק צעד אחד אחורה.

- ***look − ahead & look − back***:

look − ahead: בכל פעם שנרצה לשים ערך אנו נסתכל כיצד צעד זה ישפיע על ההמשך. כלומר - ברגע שנציב משתנה, נוכל למחוק משאר הגרף את המשתנים שלא נוכל לשים, בגלל האילוצים של הבחירה הראשונית.

look − back:כאשר נתקענו ואנו רוצים לחזור אחורה, נחשוב כמה טוב לנו לחזור אחורה האם צעד אחד או יותר.

4 שבוע 4 - *Knowledge Representation*:

4.1 הרצאה 4:

- ***Knowledge Representation***: השתמשנו באלגוריתמי חיפוש כאשר ידענו מהו מבנה הבעיה, והשתמשנו במבנה זה כדי ליצור אלגוריתם יעיל יותר לפתרון הבעיה.

כעת, אנו נלמד כיצד לשמור בעיות במבני נתונים מסויימים, בכדי שנוכל לגשת אליהן יותר בקלות. למעשה זהו ייצוג

מידע כך שמחשב יוכל לעשות בו שימוש.
אנו נשמור את המידע ($facts$) ולפי המידע אנו נחליט כיצד להתנהג במצבים מסויימים, זה מצב יותר כללי, מאשר אלגוריתם חיפוש שיודע לעשות משהו ספציפי.

- $\boldsymbol{Knowledge\ bases\ (KB)}$: זהו כמו מבנה נתונים שמאכסן בתוכו את המידע על העולם בשפה פורמלית. ואחכ בהתאם למצבים האלגוריתם יודע מה צריך לעשות וכיצד להגיב.
למעשה בסיס הנתונים שלנו מתעדכן בכל פעם על סמך החלטות שלקחנו.

- **מה האלגוריתם חייב לדעת לעשות:**
1: לייצג, מצבים, פעולות ועוד..
2: לשלב תפיסות חדשות.
3: לעדכן מצבים פנימיים של ה"עולם".
4: להסיק מאפיינים נסתרים על העולם. (לדעת שאם יש גשם אזי יהיה בוץ)
5: להסיק איזו פעולה מתאימה.

- **לוגיקה:** שפה רשמית לייצוג המידע כך שנוכל להסיק ממנו מסקנות (יכולה להיות גם שפה אריתמטית).

- **סינטקס:** מייצג את המשפטים בשפה.

- **סמנטיקה:** מייצגת את הכוונה של המשפטים (כלומר - באילו מצבים הם חוקיים ומה הכוונה משפט חוקי). - לתת משמעות לסימנים.

- **הורשה\השלכה** $Entailment$: כל צעד מושפע מהצעד שלפניו, נאמר ש $KB = \alpha$ אמ"מ אחד נכון שהשני נכון בשני הכיוונים.
למעשה זוהי מערכת יחסים בין משפטים המבוססת על סמנטיקה.

- **מודלים** $M = models$: נאמר ש m הוא מודל של משפט α, אם α הוא נכון ב m. (מודל הוא כל הצבה אפשרית של המשתנים המשפט)
$M(\alpha)$: היא הקבוצה של כל המודלים של α.
$KB = \alpha$ אמ"מ $M(KB) \subseteq M(\alpha)$

- $\boldsymbol{model\ checking}$: ראשית ננתח את החוקים שיש לנו בבסיס הנתונים + המסקנות שהסקנו = KB. אח"כ ניצור לנו α (מצב שבו נראה להתקדם) ונראה האם הוא מתמשך עם KB לפי החוק של המודלים, ואם כן - נתקדם.

- **הסקה -** $\boldsymbol{inference}$: $KB \vdash_i \alpha$ משפט α הוא נגזרת של KB לפי הפרוצדורה i.
כלומר - בעזרת i נוכל להסיק את α מ KB. נוכל להסיק בעזרת נאותות ושלמות (מוגדרים למטה):

- **נאותות\תקיפות** -$\boldsymbol{soudness}$: i תקף, אם $KB \vdash_i \alpha$ מתקיים, אזי זה נכון שגם $KB \models \alpha$.
נאמר שמערכת סינטקס היא נאותה אם כל דבר שניתן להוכיח בה - תמיד יוצא נכון ללא קשר לסדר המשתנים.

- **שלמות** $\boldsymbol{completeness}$: i שלם אם $KB \models \alpha$ מתקיים, אזי זה נכון שגם $KB \vdash_i \alpha$.
נאמר שמערכת היא שלמה - אם כל דבר שהוא נכון, הוא גם בר השגה.

- **תחשיב פסוקים** - $PL = propositional - logic$: באותו האופן שהשתמשנו בסימנים לוגים עד עכשיו $\neg, \lor, \land$
(אולי נשתמש גם ב $\Rightarrow, \iff$). מתחת נמצא טבלת אמת - $truth\ tables$
המערכת הזו היא גם נאותה וגם שלמה.

Truth tables for connectives

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

- **שימוש:** נציב את ההסקות והחוקים לפי מיקומים בטבלת אמת, ונפענח לפי הטבלה לאילו מיקומים אנו יכולים ללכת ולאילו לא.
נסתכל על KB (בסיס הנתונים) ונראה אילו הצעות יכולות להתאים למה שכתוב בעמודה של KB בטבלת האמת.
מספר האפשרויות הוא 2^n, עבור n מספר האפשרויות.

- **תקיפות** - $validity$: דרך נוספת מלבד הוכחות, כדי להוכיח שמשפט הוא נכון.
נאמר שמשפט a הוא **תקף**, אם הוא נכון **בכל** המודלים - (מודל הוא כל הצבה אפשרית של המשתנים).
הקשר להסקה הוא: $\alpha \models KB$ אמ"מ $KB \Rightarrow \alpha$ תקף.

- **ניתן לסיפוק** - $satisfiability$: נאמר שמשפט a הוא **מרצה**, אם הוא נכון **בחלק** המודלים, - קיים מודל שבו הוא נכון.

- **לא ניתן לסיפוק**- $unsatisfiability$: נאמר שמשפט a הוא **לא ניתן לסיפוק**, אם הוא **לא נכון באף מודל**.
הקשר להסקה הוא: $\alpha \models KB$ אמ"מ $KB \land \neg\alpha$ לא ניתן לסיפוק.

- **שיטות הוכחה:** מתחלקות לשתי סוגים:
1 כללי היסק - $inference\ rules$: הוכחת משפט חדש ממשפט קיים.
2 $model\ hecking$: טבלת אמת. שימוש בבק־טרקינג והיוריסטיקות חיפוש. נעבור על כל ההצבות האסשריות במשפט, ונבדוק אם בכולן המשפט יוצא $true$. אם כן - המשפט נכון.

- $Forward\ chaning$ **אלגוריתם להוכחה**: הרעיון הוא להיפטר מכל החוקים ב KB שמוסכמים רק על חלק מהמודלים ($satisfiability$). ולהוסיף ל KB את המסקנות עד שנמצא את השאילתה. כלומר - נבסס את ההוכחה שלנו על כמה משפטים שהם עובדות, ונתקדם כך עד הפתרון.
האלגוריתם רץ בזמן לינארי. נשתמש בו כאשר התוצאה לא ידועה - נעבוד לפי הסכימה ונמצא את הפתרון.
הוא אלגוריתם מושלם ונאות, אך אנו צריכים לאסור שלילות ב KB. אם לא ניתן להוכיח - האלגוריתם יקלע ללולאה אינסופית משום שהבעיה היא אינסופית.

- $\boldsymbol{Backward\ chaning}$ **אלגוריתם להוכחה**: עובד באותה השיטה של שרשור קדימה, רק הפוך. אנו מתחילים מהדבר היחיד שאנו רוצים להוכיחת וחוזרים אחורה.

  נוסיף לרשימה את כל הדברים שטרם הוכחנו, ובכל שלב נבדוק האם נכשלנו או הוכחנו כל משפט מהרשימה - כך עד הסיום. האלגוריתם רץ בזמן לינארי אך יכול לרוץ בזמן פחות בהרבה, ביחס לגודל של KB.

- $\boldsymbol{Rosolution}$: דרך נוספת של כללי היסק. אנו מקבלים כמה משפטי or שכל אחד מהם בפני עצמו לא מקדם אותנו, אך שניהם יחד יכולים להסיר במקצת את הספק. ומסיקים מהן עובדות חדשות שיכולות לקדם אותנו לפתרון.

- קיימים עוד שני אלגוריתמים ל $propositional\ inference$ על ידי $model\ cheacking$.הם אלגוריתמים שמספקים נוסחה - הצבה מספקת אחת, (מתאימים לפתרון בעיות $planing$). האלגוריתמים הבאים:

- $\boldsymbol{DPLL-Algorithem}$: זהו אלגוריתם בק־טרקינג מושלם שמשתמש בהיוריסטיקות בכדי לחסוך חזרות אחורה.

  הוא עובד באופן הבא:

  1: עבור כל טענה אם חלק ממנה לא נכון -פוסלים את כולה, ואם כולה נכונה -מאשרים אותה.

  2: אם יש לנו מספר טענות, ויש לנו תנאי שבחלק מהטענות הוא מגיע ישר ובחלק בצורת שלילה, אזי נתייחס אליו כתנאי לא טהור.

  3: אם יש רק תנאי אחד בסעיף אזי הוא חייב להיות אמת. ואם יש לפניו סימן שלילה אזי הוא שלילי (כי הוא חייב להיות תנאי חיובי בפני עצמו, ושלילת שלילי זה חיובי).

  4: אם יש טענה עם תנאי or ואנו יודעים שחלקה נכון, אזי מתייחסים לכולה נכונה ונמחק אותה.

- $\boldsymbol{WalkSAT\ Algorithem}$: זהו אלגוריתם לא מושלם לחיפוש מקומי.

  אנו בוחרים להציב תנאים להיות אמת או שקר באופן רנדומלי.

  לאחר מכן נבחר טענה אחת להיות טענת $false$, ונהפוך את התנאי הרנדומלי שנתנו לתנאים של הטענה שבחרנו.

  אח"כ נהפוך את התנאי שיחזיר לנו $true$ עבור הכי הרבה טענות.

- בבעיות קטנות נשתמש ב $DPLL$ ובבעיות גדולות ננבה להשתמש ב $WalkSat$ ואם לא נצליח לפתור נעשה טרנפורמציה לבעיה ונרית עליה שוב $WalkSat$.

4.2 תרגול 4:

- **בעיות סיפוק אילוצים - $\boldsymbol{CSP}$ הגדרות:**

  $\boldsymbol{Unary\ Constraint}$**:** אילוץ על משתנה יחיד - בד"כ אילוץ שמשתנה חייב להיות ערך מסויים, אז פשוט נציב את הערך.

  $\boldsymbol{Binary\ Constraint}$: בעיקר נדבר על זה, אילוץ על שני משתנים חייב להתקיים יחס ביניהם.

  $\boldsymbol{Binary\ CSP}$:אילוצים רק על משתנה אחד או שניים.

- $\boldsymbol{Game\ Playing}$**:** משחק סכום 0, אם ניצחתי קיבלתי 1, אם הפסדתי קיבלתי 1-. המטרה בכל שלב היא להישאר עם רווח חיובי.

  ייצוג: בכל צומת יש מצב של המשחק, העלים הם הפסד. כל שחקן נותן ערך לעדים אם ניצחנו או הפסדנו, וההנחה

היא שכל שחקן מנסה למקסם את הרווח שלו.
אנו נסתכל על כל המהליכיים העתידיים ונראה באילו מהם אנו יכולים לנצח.

- $\boldsymbol{MinMax\ Algorithem}$: אלגוריתם רקורסיבי. אנו נותנים ערך לכל צומת, באופן הבא:
אם זה צומת של המשחק הסתיים - הערך הוא למנצח.
אם זה $max\ node$ - אנו נקח את המקסימום מבין הערכים של הילדים.
אם זה $min\ node$ - אנו נקח את המינימום מבין כל הילדים.

- $\boldsymbol{MinMax\ Algorithem\ Properties}$:
הוא **מושלם** כי תמיד נגיע לסוף המשחק.
הוא **אופטימלי** רק נגד יריב אופטימלי.
סיבוכיות זמן: $O(b^m)$.
סיבוכיות מיקום: $BFS = O(b^m)$, $DFS = O(b \cdot M)$.

- **כיצד נשדרג את** $\boldsymbol{MinMax\ Algorithem}$:

- $\boldsymbol{\alpha - \beta\ Pruning}$: הרעיון הוא להסתכל על השלב של שחקן המינימום ולראות אילו בחירות הוא יבצע, ולבחור את הבחירות שלנו כך שנשאיר לשחקן המינימום אופציות פחות טובות מבחינתו. לכן אם נגיע למקרה שהוא הטוב ביותר עבורנו, נבחר אותו ונוכל להפסיק את החיפוש. לאחר מכן נכליל את החיפוש לכל רמות העץ.
פורמלית: α: הערך הכי טוב שראינו עבור שחקן המקסימום. β: הערך הכי טוב שראינו עבור שחקן המינימום. ובכל זמן אנו נחתוך לפי **ההפוך.**

- $\boldsymbol{\alpha - \beta\ Properties}$:
ה $pruning$ לא משנה את התוצאה הסופית. אלא הוא רק מקצר את זמן הריצה.
הסדר שבו אנו בודקים את הצמתים, **מאד משנה.**
אם יש לנו ערך סידור מושלם" סיבוכיות הזמן היא $O(b^{\frac{m}{2}})$
במקרה הגרוע ביותר - אין שום שיפור.

- **אלגוריתם** $\boldsymbol{Epectingmax}$: אם אנו לא בטוחים ששחקן המינימום אכן יבחר את הערך המינימלי , אלא הוא בוחר מהלכים אקראיים - אנו נחשב את תוחלת הבנים של כל מצב, ונבחר את המצב שבו תוחלת הבנים גבוהה יותר.

5 שבוע 5:

5.1 הרצאה 5 - $first\ order\ logic$:

- $\boldsymbol{Hard\ satisfiability\ problem}$:
m: מספר הסעיפים ($clauses$) - $(A \wedge B)...$.
n: מספר הסמלים ($symbol$) - A, B, C.
נבדוק את היחס $\frac{m}{n}$- ככל שהוא יותר גבוה - לבעיה יש יותר מגבלות.
בעיה קשה תתקרב ליחס של $\frac{m}{n} = 4.3$.

- **יתרונות וחסרונות של** $propositional\ logic$**:**
  1: הצהרתי ־ $declarative$: אנחנו יכולים לחבר חלקים מסויימים ולהגיע לעובדות נכונות.
  2: מאפשר הצהרה או שלילה חלקית.
  3: מוכל ־ $compositional$: הנכונות של משפט מסויים מוכלת בנכונות של משפט אחר גדול יותר.
  4: לא תלויה בניסות.
  5: חסרון ־ יש לה כח ביטוי מוגבל.

- **לוגיקה מסדר ראשון ־** $\boldsymbol{first\ order\ logic}$**:** בדומה ל $propositional\ logic$ אך זו לוגיקה חזקה יותר. אנו נראה כיצד אנו יכולים להתאים רעיונות מ $propositional\ logic$ ולהסיק בהגיון ללוגיקה מסדר ראשון.

- **ההבדל בין** $\boldsymbol{first\ order\ logic}$ **ל** $\boldsymbol{propositional\ logic}$**:** כאשר $propositional\ logic$ מניחה שהעולם מכיל עובדות מסויימות, $first\ order\ logic$ מניחה את הדברים הבאים בנוסף:
  1: **אובייקטים ־** אנשים, בתים, מספרים, תיאוריות וכו...
  2: **מערכות יחסים ־** צבע, גדול מ, אח של...
  3: **פונקציות ־** אבא של, אחד יותר מאשר וכו...

- **אמת ב** $first\ order\ logic$**:**
  1: משפט הוא $true$ אם הוא מכבד את המודל ואת הפרשנות.
  2: מודלים מכילים אובייקטים (אלמנטים מהדומיין) ומערכות יחסים בניה.
  3: הפרשנות מפנה מפורשות באופן הבא לעולם האמיתי: סמלים קבועים ־ אובייקטים, סמלי פונקציות ־ יחסים בין פונקציות.
  4: מספט אטומי נכון, אמ"מ האובייקטים שמפנים לתנאים ביחסים תואמים.

- **תנאי לכל** $\forall$**:** נאמר שמשפט נכון, אם כל תנאי ותנאי במבחן הוא נכון.

- **תנאי קיים** $\exists$**:** נאמר שמשפט נכון אם קיים תנאי שנכון ־ לפחות תנאי אחד קיים ונכון.

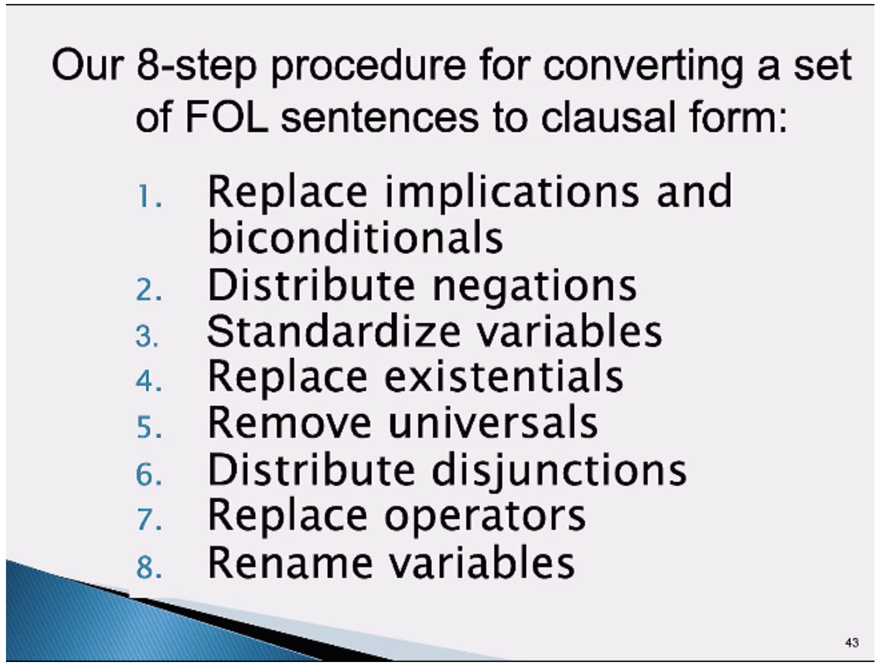
- **תכונות של תנאי לכל וקיים:**
  1: $\forall x, \forall y = \forall y, \forall x$
  2: $\exists x, \exists y = \exists y, \exists x$
  3: **לא מתקיים שוויון** $\exists x, \forall y \neq \forall y, \exists x$
  4: $\exists x, \forall y\ Loves(x, y)$ **־** קיים אדם בעולם שאוהב את כולם.
  5: $\forall y, \exists x\ Loves(x, y)$ ־ לכל אדם בעולם יש לפחות אדם אחד שאוהב אותו.
  6: **שלילת תנאי לכל:** $\forall x\ Loves(x, iceream) = \neg\exists x\ \neg Loves(x, iceream)$
  7: **שלילת תנאי קיים:** $\exists x\ Loves(x, iceream) = \neg\forall x\ \neg Loves(x, iceream)$

- $\boldsymbol{Rrducing\ contd}$**:** ניתן להפוך משפטים מ KB של $first\ order\ logic$ ל $propositional\ logic$. קיים אלגוריתם לפתרון הבעיה וכיצד להפוך בעיות מבסיס נתונים אחד שני.

1: $\boldsymbol{Unification}$**:** עבור כל משפט נמצא את החלק התואם לו במשפט שמולו ־ $knows(jhone, x) \wedge knows(jhone, jane) \Rightarrow$ $\{x \backslash jane\}$. אנו לא נאפשר לתנאי להיות גם בצד ימין וגם בצד שמאל.

2: $MGU - most\ general\ unifier$**:** בכל פעם אנו נחפש את המאחד הכללי ביותר - $P(A,Y,Z) \wedge P(X,Y,Z)$ אזי נסיק ש $\{X\backslash A\}$

- **אלגוריתם כיצד להפוך משפט מ** FOL **ל** $clause\ form$:

- 

4: $replace\ existentials$: אנו רוצים להיפטר מכל כמתי הקיים. נתון $\forall y \exists x P(x,y)$, הזהות של x שהופכת את הביטוי לנכון, תלויה בערך של y. לכן ניתן להחליף את x בפונקציה של y כך: $\forall y P(G(y),y)$

הערה: G נקראת פונקציית שקולם, ולכל כמת אני צריכים להגדיר פונקציה נפרדת.

5: $remuve\ universals$**:** נזרוק כל כמת $\forall$, ונניח שכל המשתנים הם תחת כמת של $\forall$.

5.2 תרגול 5 $Resolution$:

- $Resolution$ שיטת הוכחה נוספת. נצטרך לעבוד בשיטת $CNF - conjuction\ normal\ form$. שיטת ההוכחה היא - אנו מסיקים על ידי הוכחה בשלילה. לדוגמא אם יש לנו שני סעיפים $p \vee q$ ו $\neg p$ אנו יכולים להסיק מהם את q . שיטה זו היא מושלמת ונאותה.

- המרת $resolution$ ל CNF:

1: החלפת גרירה דו כיוונית בשתי גרירות חד כיווניות

2: $\alpha \Rightarrow \beta = \neg \alpha \vee \beta$

3: חוקי דמורגן - שלילה לפני סוגריים

4: חוק הפילוג.

6 שבוע 6:

6.1 הרצאה 6:

- ***unifiable*- בלתי ניתן לאיחוד:** נאמר שקבוצה של ביטויים $\{\Phi_1, ..., \Phi_n\}$ **אינה ניתנת לאיחוד**, אם קיימת תמורה σ שמשווה בניהם ע"י כפל: $\Phi_1\sigma = ... = \Phi_n\sigma$.

6.1.1 $Planning$:

- **מוטיבציה ל $planning$:** כאשר אנו רוצים לייצג בעיה מסויימת אנו צריכים לייצג נקודת התחלה, מטרה, אוסף של אופרטורים שעוזרים לנו להתקדם מהמצב הנוכחי למצב הבא, **אין** לנו היוריסטיקה מפורשת.
אנו צריכים ליצור תכנית מטרה כללית - $general\ purpose\ program$ שתתן לנו תיאור בשפה סטנדרטית, ותחזיר רצף פעולות של אופרטורים, שיכווינו אותנו כיצד להגיע מנקודת ההתחלה אל המטרה. כלומר - התכנית תחזיר לנו מסלול כיצד להגיע אל המטרה.

- **$Plans\backslash solutions$ - תכניות:** תכניות הם אוסף של מהלכים העוזרים לנו להגיע מנקודת ההתחלה אל נקודת הסיום, לא כל התכניות רצויות באותה מידה.

- **מה המשימות שלנו:** למצוא האם יש פתרון, למצוא פתרון כלשהו, למצוא פתרון אופטימלי, ועוד...

- **ההבדל בין $Scheduling$ ל $planning$:** בעוד $Scheduling$ מחליט **מתי** לבצע סט של פעולות **נתונות**, $planning$ מחליט **מה** הפעולה שצריך לבצע (**ומתי**), בכדי להשיג את המטרה.

- **השילוש הקדוש של AI:** $models$ - כדי להגדיר ולהבין את הבעיה. $languages$ - כדי לייצג את הבעיה. $algorithems$ - כדי לפתור את הבעיה.

- $planning$ הינה צורה של פותר בעיות כללי, כמו השילוש הקדוש. אנו מכניסים בעיה ושפה ואז ה $planning$ מוציא לנו פתרון.

- $planning$ עובד עם בסיס נתונים בדומה ל FOL, אך בניגוד אליה יש לו גם את היכולת להסיר דברים לא רלוונטים מה KB.

- **מודל של** $classical\ planning$ (סוג אחד של $planning$):
S: מצבים.
$s_0 \in S$: נקודת ההתחלה.
$S_G \subseteq S$: קבוצת נקודות הסיום.
$A(s) \subseteq A\ for\ s \in S$: פעולה ישימה.
פונקציית מעבר: $s' = f(a, s)$ for $a \in A(s)$.
פונקציית עלות: $A^* \rightarrow [0, \infty)$.
פתרון: הוא אוסף של פעולות שניתנות ליישום המובילות אותנו מ s_0 ל S_G.

פתרון אופטימלי: c.
$classical\ planning$ הוא דטרמיניסטי, ושימושי לבעיות פשוטות בלבד.

- **הגדרה - $transition\ system$:** היא הקבוצה הבאה $\langle S, I, \{a_1, ..., a_n\}, G\rangle$ כאשר:
  S: קבוצת מצבים.
  $I \subseteq S$: קבוצה של מצבים התחלתיים.
  $a_i \subseteq S \times S$: כל פעולה a_i מגדירה יחסים בינארים על הקבוצה S.
  $G \subseteq S$: קבוצת נקודות הסיום.

- **הגדרה - פעולה חוקית\ישימה:** פעולה a היא חוקית במצב s אם היא מקדמת אותנו למטרה לפחות במצב אחד.

- **הגדרה - $deterministic\ transition\ system$:** מערכת מעבר דטרמיניסטית רק כאשר יש לנו נקודת התחלה **אחת**, וכל הפעולות הן דטרמיניסטיות.
  היא הקבוצה הבאה $\langle S, I, O, G\rangle$ כאשר:
  S: קבוצת מצבים.
  $I \in S$: מצב התחלתי.
  $a \in O$ $(\text{with } a \subseteq S \times S)$: פעולה a היא פונקציה חלקית.
  $G \subseteq S$: קבוצת נקודות הסיום.

- **הגדרה - $successor\ state$:** אם הגענו מהמצב s למצב s' בעזרת הפעולה a, נסמן $s' = app_a(s)$.

- **הגדרה - $plan$:** תכנית ל $\langle S, I, A, G\rangle$ היא סדרה של פעולות $\pi = a_1, ..., a_n$.
  נסמן זאת כך: $app_{a_n}\left(app_{a_{n-1}}\left(\ldots app_{a_1}(I)\ldots\right)\right) \in G$.

- **ייצוג מטריציוני של מערכת מעבר:**

עבור n מצבים נייצג מטריצה של $n \times n$, השורות מייצגות את נקודת ההתחלה והעמודות את היעד. וכל תא מייצג האם ביצענו פעולה, נסמן 1 כאשר ביצענו פעולה, ו 0 אחרת.
כאשר יש לנו כמה מטריצות שכל אחת מהן מייצגת מספר פעולות ארות, ניתן לכפול אותן יחד ולגלות אילו פעולות במטריצה A יובילו אותנו לנקודות במטריצה B.

6.2 תרגול 6 - לוגיקה מסדר ראשון:

- **לוגיקה מסדר ראשון:** שונה מלוגיקה פסוקית בדברים הבאים - אופרטורים $\Leftarrow, \Rightarrow$ וכמתים $\forall, \exists$ שיחברו אותנו מעולם הסינטקס אל העולם האמיתי, ופונקציות שמחזירות $true\backslash false$, בתוך הפונקציה נשים משתנים מהעולם, (יש גם פונקציות שלא מקבלות פרמטרים - נתייחס אליהן כקבועים).

- **הגדרות:**
  1: נאותות - $Soundness$: נאמר שהמערכת נאותה אם לא ניתן להוכיח בה דברים לא נכונים.
  2: שלמות - $Completness$: נאמר שמערכת היא שלמה אם ניתן להוכיח בה כל משפט שנכון.

- **נאותות ושלמות ב FOL:** לוגיקה מסדר ראשון היא גם נאותה וגם שלמה. (אך כל מערכת שהיא מספיק חזקה יכולה ליצור פרדוקסים שיובלו לחוסר שלמות - משפטים נכונים שלא ניתן להוכיח). לכן לוגיקה מסדר ראשון שלמה כל עוד אין פונקציות שמקבילות לחיבור וכפל.

- **הגדרות:**
1: משפט אטומי: משפט שאין בו חיבורים לוגים בכלל (פונקציות).
2: סעיף: איווי - $\vee$ של אטומים או השלילה שלהם.
3: צורת סעיף $\boldsymbol{clause\ form}$: גימום $\wedge$ של איווים $\vee$. (כל משפט בלוגיקה מסדר ראשון ניתן להפוך למשפט בצורה הזו).

- **החלפה -** $Subtitution$: אנו יכולים להשתמש בשמונת הצעדים ולהחליף כל משפט מלוגיקה מסדר ראשון ב $clause\ form$, אך עדיין יהיו לנו משתנים. לכן נשתמש בהחלפה - החלפה של משתנים במשתנים אחרים (כדי שנוכל לצמצם), או החלפה של משתנה באובייקט ממשי.

- **איחוד של ביטויים -** $Unification$: החלפה שגורמת לשני ביטויים להיראות אוו הדבר. נעשה זו בעזרת פונקציה $Unify(\alpha, \beta) = s$ שתתאם לנו החלפה לכל שני ביטויים.
משתנים ברי איחוד - $unifable$: נאמר ששני משתנים הם ברי איחוד אם ורק אם יש החלפה שמאחדת אותם - גורמת להם להיראות אותו הדבר. (יש כמה החלפות שמקיימות את הדרוש ולא רק אחת).
$\boldsymbol{MGU - most\ general\ unifier}$: ההחלפה הכללית ביותר שתסגור לנו כמה שפחות אופציות בהמשך. (קיים אלגוריתם שמבצע את הדרוש).

- $\boldsymbol{Resolution}$: אנו לוקחים שני ביטויים ומסיקים מהם על אחד מהסעיפים, ואחכ משליכים הלאה, לבסוף נגיע לכמה סעיפים שאנו יודעים שכמה מהם נכונים.
הסקה - $Derivation$: אם אנו רוצים להוכיח ש β נובע מ $\alpha_1....\alpha_n$ נסמן זאת כך $\alpha_1....\alpha_n \vdash \beta$, והדרך להוכיח היא להניח את השלילה של β להגיע לביטוי ריק - סתירה.
$first\ order\ resolution$ היא נאותה ושלמה.

- $resolution$ **וחיפוש:** ניתן להתייחס לבעיה כאל בעיית חיפוש, אנו מתחילים מהעלים שזה בסיס הנתונים ורוצים למצוא את השורש.

- **היוריסטיקות:** אנו רוצים לצמצם את החיפוש שלנו ולא להסיק דברים לחינם שלא יעזרו לנו להוכיח את שאנו רוצים. לכן נשתמש בהיוריסטיקות, אלו אותן היוריסטיקות שהשתמשנו בהן ב $DPLL$. ועוד...
יש לנו כמה סוגים של $resolution$ שמצמצמות לנו את העץ בכדי שלאגנתפרש סתם למקומות מיותרים.

7 שבוע 7 - $Planning$:

7.1 הרצאה 7:

- $\boldsymbol{Reachability}$: אנו מתעסקים בייצוג מטריציוני של מערכת מעבר. בכל שלב אנו נסתכל על הקדקודים אליהם

אנו יכולים להגיע ב i צעדים עבור $i \in [n]$. כלומר - בכל פעם נכפול את המטריצה בעצמה ואז נשיג את הקודקודים אליהם אנו יכולים להגיע בעזרת i קודקודים אחרים. אם קיים מסלול כזה, נחבר את הקודקודים בצלע באופן ישיר.

- **שפה לייצוג בעיות:** אנו נייצג כל $state$ בעזרת משתנים, שאותם נוכל לייצג בסופו של דבר בעזרת משתנים בוליאנים.

- ***Deterministic planning tasks***: זו רביעיה $\Pi = \langle V, I, A, G\rangle$, כאשר:

  V: מייצג קבוצה של $state\ variables$ - ייצוג של מצבים בעזרת משתנים.

  I: מייצג את נקודת הסיום בעזרת V.

  A: מייצג קבוצת פעולות למעבר ממצב למצב.

  G: מייצג אילוץ\נוסחה המתארת את המטרה בעזרת V.

- **איך נעבור מ** $Deterministic\ planning$ **ל** $transition\ system$**:** $\Gamma(\Pi) = \langle S, I, A', G'\rangle$כאשר:

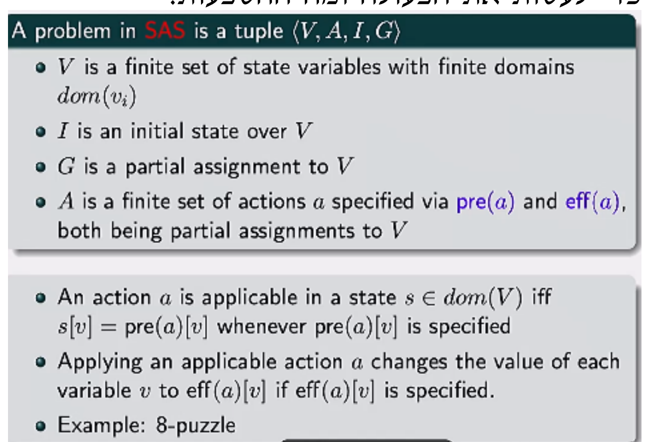
  S: קבוצה של כל ההערכות של V.

  A': $\{R(a) : a \in A\}$ כאשר $R(a) = \{(s, s') \in S \times S \mid s' = app_a(s)\}$.

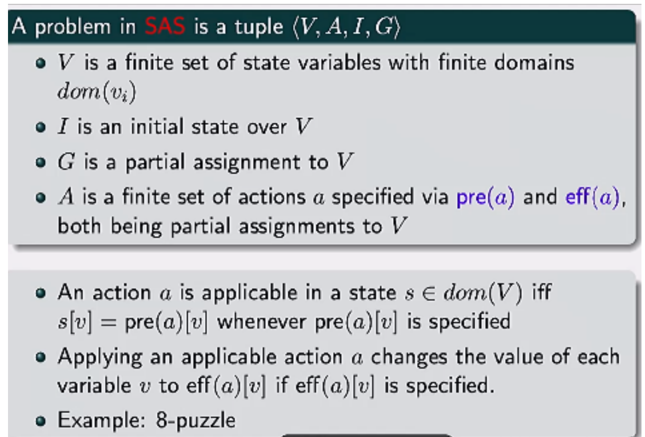
  G': $\{s \in S : s \vDash G\}$.

- ***Planning language***: שתי שפות לייצוג בעיות $planning$.

SAS: ייצוג שבו יש משתנים שלכל אחד יש ערכים שהוא יכול לקבל, ופעולות שאומרות לנו מה המצב שחייב להיות כדי לעשות את הפעולה ומה ההשפעות.



STRIPS: במקום ערכים יש אטומים של $true\backslash false$, והמצב ההתחלתי הוא תת קבוצה של האטומים האפשריים שהם $true$ וכל השאר $false$.



- ***planning algorithem***:

1 האלגוריתם יעבוד באופן הבא: בכל שלב אנו נבדוק את המרחק מנקודת ההתחלה. ובכל שלב נבדוק כיצד להגיע למצב הנוכחי על ידי צעד אחד, אחכ על ידי שני צעדים וכן הלאה.

2 *State space search*:

ראשית נבחר כיצד אנו רוצים להתקדם - מנקודת ההתחלה לנקודת הסיום, או להיפך.

regression: אלגוריתם שמתחיל מנקודות הסיום, וחוזר אחורה עד שמגיעים אל נקודת ההתחלה. בכל נקודה אנו מסתכלים מהיכן ניתן להגיע אליה, ונבחר את הצעד הטוב ביותר.

כיצד נממש ב $STRIPS$: נסמן את המצבים כ $true$ ובכל שלב שנתקדם נפסול מצבים מסויימים ונסמן אותם כ $false$, ונוסיף מצבים אחרים ונסמנם כ $true$.

- ***Planning via SAT***:

1: נעביר את בעיית התכנון ל SAT.

2: ניצור CNF שמרצה $(satisfiable)$ אמ"מ קיימת תכנית עם b צעדים שמרצה את המטרה. $(b = t_0\ To\ t_{max})$

3: נשתמש ב $DPLL$ או $WalkSAT$.

4: נחזיר את התכנית אם הצלחנו. אחרת נעלה את הערך של b ב 1.

בכל שלב שנבצע צעד נבדוק מה ההשלכות שלו על הצעד הבא.

- ***Plan Space***: רעיון שאומר במקום לשים בכל קודקוד מצב, נשים בקודקודים פעולות. ואז בכל שלב הצלעות יעבירו אותנו מפעולה לפעולה, קודקוד ההתחלה יהיה $null$. וקודקוד הסיום תהיה הפעולה המסיימת את המהלך. היתרון הוא שאנו לא אמורים לעבוד בסדר מסויים, אלא רק לבצע פעולות שיובילו אותנו אל נקודת הסיום.

- ***Casual links***: בא לסמן לנו אילו פעולות אנו כן אמורים לבצע בסדר מסויים, משום שהן תלויות אחת בשניה.

- ***Threats***: נאמר שפעולה a היא $Threats$ את פעולה b, אם נבצע אותן הפוך אזי בסדר יהרס. כלומר - ביצוע פעולה מסויימת לפני פעולה אחרת יפגע לנו בתהליך $Casual\ links$.

7.2 תרגול 7:

- **ההבדל בין שפות התכנון:** SAS תופסת פחות מקום בזיכרון מאשר $STRIPS$.

- $\boldsymbol{Planning\ graph}$: מבנה נתונים שיעזור לנו לבנות היורסטיקות לבעיות חיפוש. בנוסף אנו יכולים להשתמש באלגוריתם $graph\ plan$ בכדי לחלץ $plan$ מהגרף.
כיצד נבנה את הגרף: רמה ה 0 זה המצב ההתחלתי, אחכ יש לנו החלפה בין רמות, ברמת הפעולות מופיעות כל הפעולות שניתן לבצע, וכל האטומים שהיו ברמה הקודמת והאטומים שנובעים מהפעולה שביצענו בשלב הקודם. בין הרמות נחבר בעזרת צלעות המחברות בין $pre\ condition$ ל $action$. בנוסף יהיו צלעות בין הפעולות למצבים שהן מוסיפות. בנוסף יש פעולות שנקראות $noOp$ והן מעבירות את ה $proposition$ מהרמה הקודמת לרמה הבאה (פעולות אלו חשובות גם כן). אנו מפסיקים בנות את הגרף אם הבניה הסתיימה - יש שתי רמות זהות, או מצאנו שכל האטומים שאנו צריכים נמצאים ברמה האחרונה - הגענו לנקודת הסיום.

- **הגדרה - אטומים סותרים** $mutually\ exlusive$**:** נאמר ששני אטומים סותרים, אם כל הדרכים לקבל אותם - כל הפעולות שמייצרות אותם הן $mutex$.

- **הגדרה -** $\boldsymbol{mutex}$**:** נאמר ששתי פעולות הן $mutex$ אם:
אחת מהן מייצרת $proposition$ שהשניה מוחקת.
אחת מוחקת $precondition$ של השניה.
פעולות שה $precondition$ סותרים.

- **כיצד לייצר היוריסטיקות מ** $Planning\ graph$**:**
$\boldsymbol{max\ level}$: הרמה הראשונה שבה כל האלמנטים של המטרה מופיעים. לא מאוד מדוייק. אדמיסבילי
$\boldsymbol{set\ level}$: הרמה הראשונה שבה כל האלמנטים של המטרה מופיעים. ואין בניהם מיוטקס. אדמיסבילי
$\boldsymbol{level\ sum}$: הסכום שבו כל הרמות מופיעות - לא אדמיסבילי.
$\boldsymbol{relax\ plan}$: האורך של כל הפעולות, מספר הפעולות שצריך - מתעלם ממיוטקס.

- $\boldsymbol{PDDL}$: שפת התכנון הסטנדרטית, התכנית ממירה את השה הזו ל $SAS\backslash STRIPS$.
כיצד השפה עובדת: מחלקת את הבעיה לשני קבצים: **קובץ דומיין** - מכיל את כל הפרדיקטים, את כל סוגי אובייקטים שנרצה. **קובץ בעיה** - בעיה ספציפית שמכילה את האובייקט עם מצב התחלתי ומצב סיום, לבעיה הספציפית.

- $\boldsymbol{Abstraction}$**:** הרעיון הוא לייצר מערכת מעברים שמתייחסת רק לחלק מהמשתנים, נפתור חלק קטן ונשתמש בזה להיוריסטיקה. בכדי לקבל היוריסטיקה אדמיסבילית אנו צריכים שהמרחקים לא יהיו גדולים.
נקח מקסימום של כמה $Abstraction$ או נחבר בניהם - תמיד זה אדמיסבילי. כשנרצה לקחת סכום - צריך לבדוק שלא קיים אופרטור $a \in A$ שמשפיע על שני משתנים שונים, ואז ההיוריסטיקה היא אדמיסבילית וקונסיסטנטית.

8 שבוע 8 - $Markov$:

8.1 הרצאה 8:

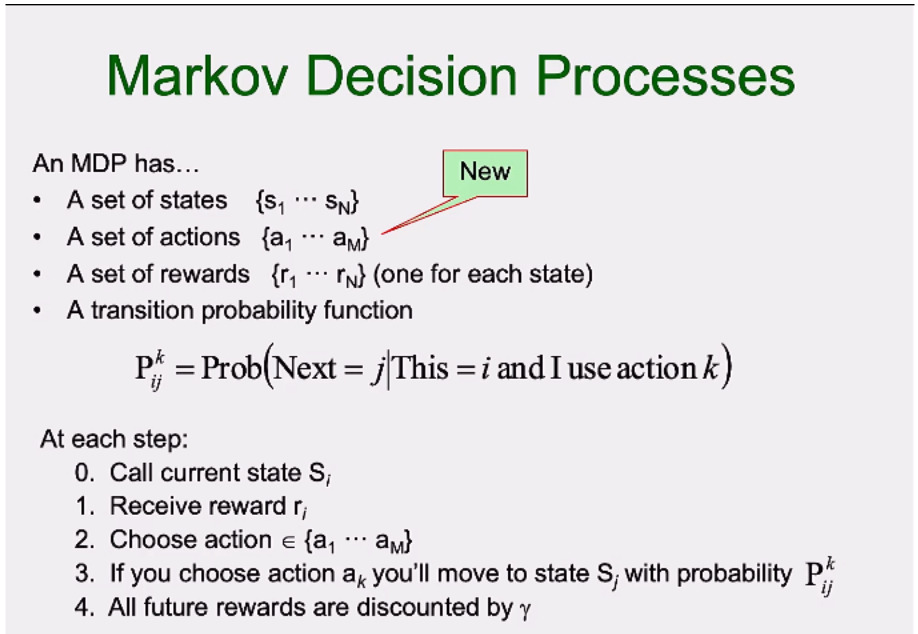
- $\boldsymbol{Markov\ system}$: שרשראות מרקוב. בכל $state$ יש לנו הסתברות עבור המעבר למצבים הבאים, לא בהכרח שכל מצב יקבל את אותה הסתברות. למעשה שרשראות מרקוב בנויות על ההנחה שכל מצב תלוי במצב הנוכחי ולא בהסטוריה שעברנו עד כה.

- $\boldsymbol{Markov\ decision\ processes - MDPs}$: הוא מודל של העולם, וההבדל ממודלים אחרים הוא שאנו מניחים שהעולם הוא לא דטרמיניסטי, כלומר - אנו לא יודעים לאיזה מצב נגיע אך יש לנו הסתברות להגיע למצב מסויים. המטרה - אנו רוצים למקסם את התועלת שלנו על ריצה אינסופית (ההנחה היא שהרבה יותר קל לנו לפתור כאשר נשאיף לאינסוף). תהליך שיעביר אותנו ממצב מסויים לשרשראות מרקוב, אותן אנו יודעים לפתור.
הוא יוגדר כטאפל $\langle S, A, \{P_{sa}\}, \gamma, R\rangle$ **כאשר:**
S: מייצג את כל המצבים בעולם, s_0 מייצג מצב התחלתי.
A: כל הפעולות האפשריות.
$\{P_{sa}\}$: $transition\,prob$: ההתפלגות על כל שאר המצבים האחרים $P_{sa}\left(s'\right) = Pr\left[\text{ getting to } s' \mid \text{ was in } s \text{ and did a }\right]$
γ: גורם הנחה - $discount\ \ factor$ אומר לנו כמה המצבים בהמשך פחות חשובים לנו - ככל שעובר הזמן הערך של המהלך יורד.
R: פונקציית תועלת $reward\ function$: פונקציה שאומרת לנו כמה שווה להיות במצב הזה (למעשה זו הפונקציה שתגדיר לסוכן מה לעשות). כאשר: $R : S \Rightarrow \mathbb{R}$

הגדרה - הילוך על בעיית מרקוב: רשימה של מצבים שעברנו בהם שיש ביניהם מעברים חוקיים (הסתברות חיובית לנוע ממצב למצב). אפשר לכלול את הפעולות אך אין צורך.
פונקציית הערכה: אנו נרצה לבצע פעולות הגיוניות שיובילו אותנו לדרך הטובה ביותר. לכן תהיה לנו $value\ \ func$, שאותה נגדיר באופן הבא: $V(s) = R\left(s_0\right) + \gamma \cdot R\left(s_1\right) + \gamma^2 \cdot R\left(s_2\right) + \cdots$.
הפונקציה הזו לא ממש טובה לכן נגדיר $Policy$.
$\boldsymbol{policy - \pi : S \Rightarrow A}$: רשימה של פעולות לעשות, שאומרת לסוכן בכל מצב איזו פעולה לעשות. נעשה זו באמצעות תוחלת, ופונקציה רקורסיבית: $V^{\pi}(s) = R(s) + \gamma \cdot \sum_{s' \in S} P_{s\pi(s)}\left(s'\right) V^{\pi}\left(s'\right)$. ניתן להגדיר את המשוואה כמטריצה ולפתור בעזרת כפל מטריצות באופן הבא: $\vec{V} = (I - \gamma[P])^{-1}\vec{r}$.
הערה: לכל MDP קיימת $policy$ אופטימלית.

הגדרה - $V*(s)$**:** התוחלת הצפויה אם נבחר את ה $policy$ הכי טוב במצב s. $\pi^{\star}(s) = \underset{a}{argmax}\left\{\sum_{s' \in S} P_{Sa}\left(s'\right) V_i^{\star}\left(s'\right)\right\}$

בשקף מופיע התהליך של MDP:

- 

- **לכן:** אנו צריכים ערך מקסימלי ואת ה $policy$ הכי טוב, וכך נוכל למצוא מסלול קיימים שני אלגוריתמים:

1: $value\ iteration$: דרך נוספת לחשב את שרשראות מרקוב, אנו בוחרים $0 < \gamma < 1$ ונכפיל בה את הממוצע בכל שלב. אנו מתחילים מ $\mathrm{J}^1(\mathrm{S}_i) = r_i$ כאשר r_i מייצג את ה $rewards$. אחכ נחשב את שלב 2 בהתבסס על שלב 1: $\mathrm{J}^2(\mathrm{S}_i) = r_i + \gamma \sum_{j=1}^{N} p_{ij} J^1(s_j)$. נמשיך באופן הזה כך שכל שלב מתבסס על החישוב של השלב הקודם.

2: $Policy\ Iteration$: מתחילים מ $policy$ אקראי לגמרי, שבוחר לבד מה לעשות בכל מצב. עד שנמצא $policy$ אופטימלי.

- **$Factored\ Markov\ decision\ processes-FMDP$**: שיפור של MDP שמאפשר לנו

8.2 תרגול 8:

- כתוב בהרצאה.

9 שבוע 9:

9.1 הרצאה 9 - $POMDP$:

- **$exploration\ VS\ exploiation$**: ההבדל ביניהם הוא כזה:

$exploration$: טיול העולם שלנו וגילוי מצבים חדשים, כך אנו מגלים מה כל מצב עושה ומה התועלת מכל מצב.

$exploitation$: אנו מסיקים מהמצבים שלנו מה התועלת ומפצעים פעולות לפי ההסקות האלה.

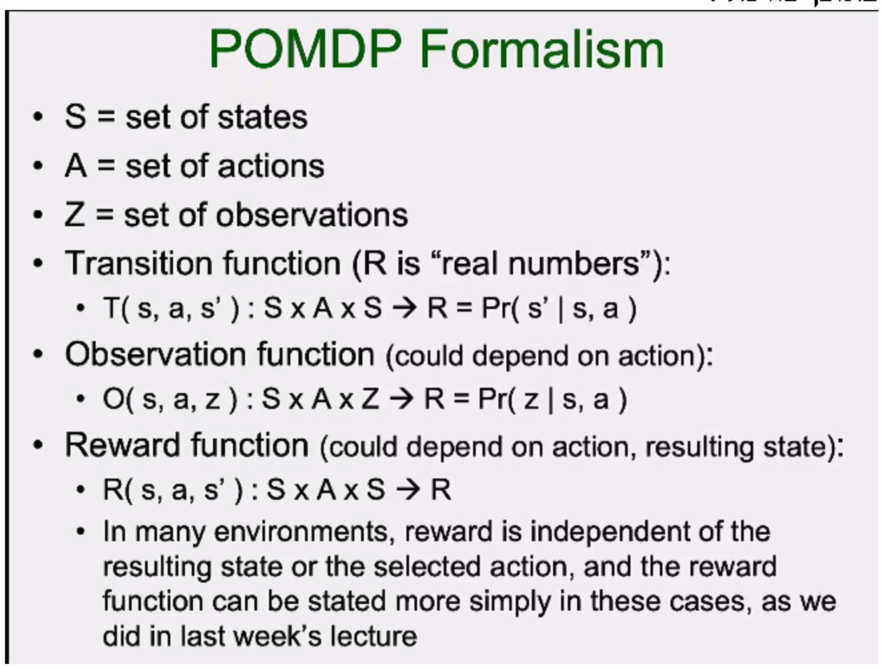
למעשה כאשר אנו לא יודעים כלום אנו נתחיל להסתובב בעולם וללמוד ע"י $exploration$, אח"כ נבצע צעדים שיקדמו אותנו למטרה שזה למעשה $exploitation$.

- ***POMDP***: יש לנו מצבים, פעולות, מעברים ופונקציית תועלת (R) בדומה ל MDP, אך הוא שונה ממנו בכך שאנו רק יכולים **לצפות** איפה אנחנו נהיה בשלב הבא, ולא יכולים לדעת בוודאות מה יקרה. אנו רואים בכל פעם רק חלק מהמצב ולא יודעים כלום על העולם, ואנו צריכים לגלות מהו מרחב המצבים. בנוסף אנו לא יודעים מה התוצאות של כל פעולה, אלא רק מהן הפעולות. אנו גם לא יודעים מה התועלת מכל מצב. (זה יותר דומה לעולם האמיתי).

אנו מוסיפים את הדברים הבאים: $observetions$ - קבוצה סופית של תצפיות אפשריות. בנוסף תהיה לנו פונקציית תצפית - $observetion\ func$ - פונקציה שמקשרת בין הפעולות לתצפיות.

הגדרה - פרק *episode*: בהינתן פעולה π, פרק הוא סדרה של פעולות, התוצאות שלהן וההישיגים הנובעים מהן.

באופן פורמלי:



הערה: אנו לא יכולים להשתמש כאן בשרשראות מרקוב, משום שאנו לא יודעים איפה אנחנו נמצאים בכל שלב, אלא אנחנו יכולים רק לנחש, לכן נשתמש באלגוריתמים הבאים:

- ***beleif state***: דרך להשתמש בשרשראות מרקוב ב $POMDP$, נתחיל עם כל המצבים ונבדוק עבור כל מצב מה התועלת היוצאת לנו ממנו. אחרי כל השיטוט נתחיל למפות עבור כל מצב את התועלת ונבדוק הסתברויות. ממצבים אלו נוכל לבנות שרשראות מרקוב.

9.2 תרגול 9 - RL:

- **למידה מחיזוקים** - ***RL − reinforcment learning***: אנו רוצים ללמד את המערכת להתקדם בכוחות עצמה.

9.2.1 אלגוריתמים לשימוש ב RL:

- ***model base***: שיטה לבנות MDP, נבנה מהעולם הנתון הסתברויות על ידי כניסה לפעולות והבנה מה יוצא מהן. החסרון הוא שכדי להעריך באמת, נצטרך לבקר מלא פעמים בכל מצב.

- ***monte carlo*:** בגדול - השיטה אומרת לנסות הרבה דברים ובסוף לקחת את הממוצע. אצלנו - אנו נעשה הערכה ל π שלדעתנו יהיה הטוב ביותר, ונשפר אותו לפי הצורך. עבור כל צעד נבדוק איך הוא השפיע עלינו, ונשפר את π בהתאם.

- **$Q - Learning$:** ע

9.2.2 רשתות נויירונים - NN:

- **רשתות נויירונים:** לומדות מדוגמאות, אנו מכניסים להן מידע ומאמנים אותן להחזיר את הקלט הנכון.

10 שבוע 10 - $Learning$:

10.1 הרצאה 10:

- ***inductiv learnung methot*:** אנו מקבלים זוג $(x, f(x))$, ואנו צריכים למצוא h שמסכימה עם f. אם מצאנו h כזו אמר שהם מתאימות - $adjust$.

  הגדרה - קונסיסטנטיות: נאמר שהן קונססטסטיות, אם h מסכימה עם f על **כל** הנקודות.

- ***overfitting*:** מתאר מצב בו אנו מנסים לענות על כמה שיותר נקודות ומקבלים פונקציה ש "קופצת", אנו מחפשים פונקציה שתענה על כמה שיותר נקודות אך לא בכל מצב. לכן לפעמים נעדיף לענות על פחות נקודות, אבל לשאוף לקו ישר יותר.

- **עץ החלטה:** עץ החלטה למעשה הוא פונקציה שמביאה לנו קלט, ואנו מחזירים כפלט את פתרון הבעיה - בוליאני $true \backslash false$.

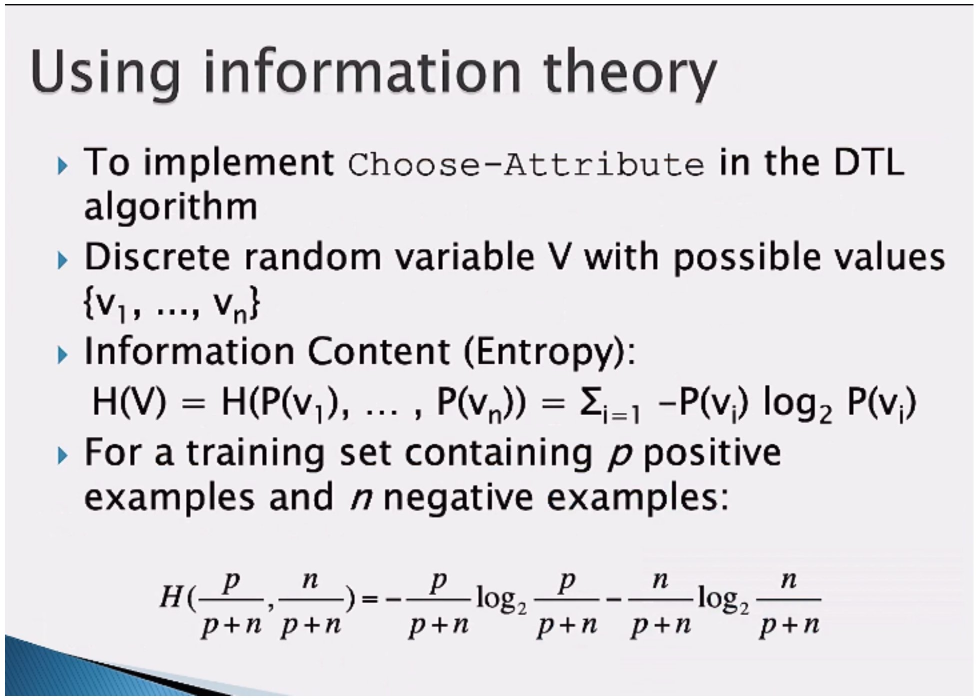
- ***Desition tree learning*:** מציאת עץ החלטה היא פעולה מורכבת, משום שמספר העצים עבור כמה תנאים הוא אקספוננציאלי בגודל הקלט. לכן נשתמש בשיטה הבאה:

  נמצא עץ החלטה קטן שעונה על הדרישות של כל הדוגמאות.

  הרעיון: למצוא באופן רקורסיבי את התכונה המשמעותית ביותר ולשים אותה בשורש.

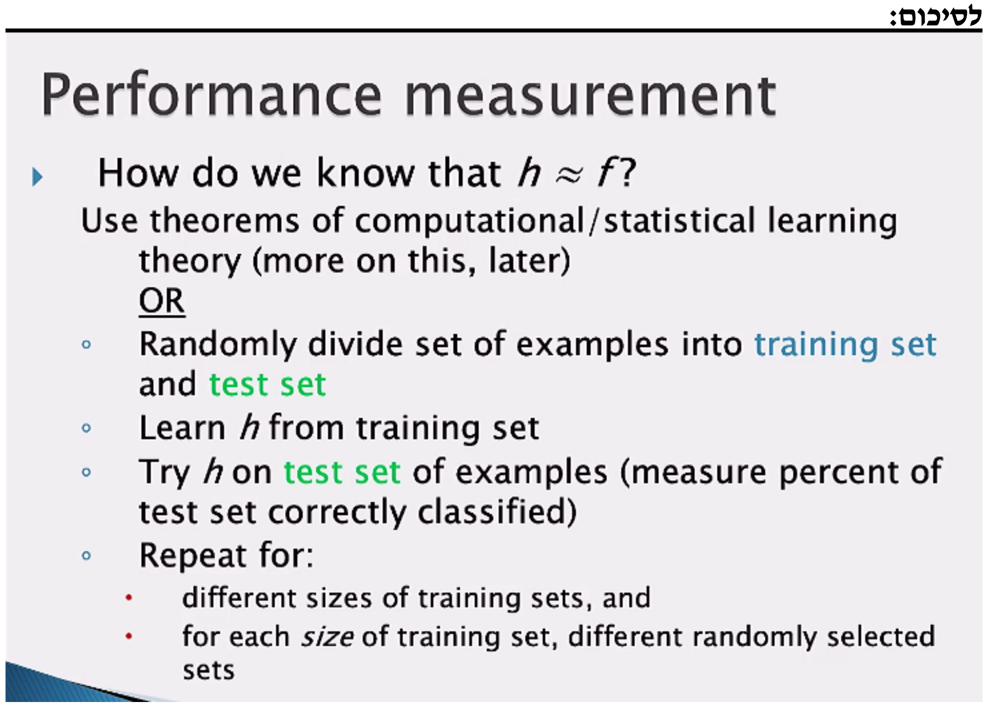
- **אלגוריתם למציאת עץ החלטה - *DTL*:** נעבוד כמו שכתבנו בסעיף הקודם. אם מציאת התכונה הטובה ביותר נבצע באופן הבא: בכל שלב נבחר את הענף שבו כל הדוגמאות $true$ או כל הדוגמאות $false$. לכן תמיד נעדיף לבחור ענף שיוביל אותנו להחלטה ולא למיקס של $true$ ו $false$.

- ***information theory*:** נשתמש בשיטה זו באלגוריתם DTL בכדי לבחור בכל פעם את הענף הטוב ביותר. בכדי לחשב נשתמש בנוסחה הבאה לחישוב H (הפונקציה H נקראת $entropy$):



- **$information\ gain$:** עבור כל ענף נחשב $H() - remainder$ ונבחר את הענף שמחזיר את התוצאה המקסימלית.

- **לסיכום:**



10.2 תרגול 10 - למידה מתצפיות:

- **סיווג:** אנו מקבלים מאורע חדש או נתונים, ואנו רוצים לחזות מה יקרה. אנו מקבלים סט של נתונים ואנו אמורים להחליט לפי הנתונים.

- **עץ החלטה:** עץ כלשהו שבו בכל צומת יש שאלה, הצלעות הן הערכים שכל נתון יכול לקבל, וכשנגיע לעלה זהו הסיווג שחיפשנו.

- **כיצד נבנה עץ החלטה:** נבחר תחילה לשים את כל הענפים שאנו מסכימים עליהם, ואח"כ נפצל לתתי עצים את כל שאר הבעיות שאין לגביהן החלטה חותכת מהנתונים. בנוסף נצטרך לבחור מתי לעצור - נעשה זאת כשנגיע להחלטה "טהורה".

  אנו נעדיף לא לבנות את העץ רנדומלית, משום שזה ייצור לנו עץ גדול מאד, ועצים גדולים הם פחות מדוייקים. לכן נבנה עץ קטן.

- **כיצד נבנה עץ קטן:** נשתמש בתורת האינפורמציה.

  1: נבחר לשאול שאלות שיחלקו את התשובות האפשריות לחצי - שאלות של כן או לא.

  2: עבור כל משתנה נבדוק מהו פונקציית ה $entropy$ שלו, ונבדוק כך איזה נתון הוא המשמעותי ביותר ונותן לנו הכי הרבה מידע.

  3: מידע משותף - I: $I(X;Y) = H(X) - H(X|Y)$ נחשב את האנטרופיה של X ונחסר ממנה את האנטרופיה בתנאי $X|Y$. נסיק מידע בדומה לסעיף 2.

  4: נבחר ללכת דרך הענף שיתן לנו הכי הרבה מידע.

- ***overfitting* בעצי החלטה:** כאשר אנו מסווגים מידע לא חשוב כמידע חשוב, ומנסים להתחשב בו גם כן בכדי לדייק בכל הנקודות. לדוגמה - האלגוריתם מחשב את מספר האוהדים המשתתפים במשחק ומחליט שזה משפיע על הנצחון, מה שכמובן לא קובע ומשפיע.

  לכן ננסה לאזן בין דיוק לפשטות, נשתמש ב $Gain\ ratio$.

- ***Gain ratio*:** בכדי להתמודד עם הבעיה של $information\ gain$ אנו נשתמש בשיטה הזו. בה אנו נכפול כל משתנה במידת החשיבות שלו, כך שמשתנים חשובים פחות יקבלו דירוג נמוך. לכל משתנה ניתן דירוג - כמה מידע רלוונטי למטרה ניתן להפיק ממנו.

  אך עץ זה הוא טוב למידע הנוכחי, אך הוא יכול לטעות עבור מידע חדש. לכן נשפר אותו כך:

- **שיפור השלב הקודם:** המטרה היא להגיע לעץ יותר קטן. נעשה זאת כך: נתחיל מעץ גדול ונצמצם אותו כך שיהיה לנו עץ קטן יותר, על ידי הורדת ענפים מיותרים. כלומר - בשלב הלימוד נאפשר יותר שגיאות, בכדי שבשלב ההחלטה נקבל עץ מושלם.

  נבצע זאת כך: $cross\ validition$ - ניתן לעץ לעבוד רק עם חלק מהנתונים וראה האם המסקנה שלו תואמת לנתונים שבידינו. כלומר - נבחן את העץ על ידי כך ש"נעלים" מידע מהתחלה ונראה כיצד העץ מחליט. כך נוכל לבדוק האם בזמן אמת העץ גם יחליט נכון על אף שאין לו את כל הנתונים. אותו הדבר נבצע כשנוריד עלים, נבדוק עבר כל עלה האם ניתן להוריד אותו ולשפר את העץ.

11 שבוע 11 - $Game\ theory$:

11.1 הרצאה 11:

- **ייצוג מטריציוני עבור** $Game$: אנו מייצגים במטריצה את כל המצבים - האסטרטגיה שכל שחקן יבחר, והתוצאה עבור כל אחד מהם עבור כל אסטרטגיה.

- **מינמקס עם ייצוג מטרציוני:** השחקן A יסתכל על השורות ויבחר את השורה שבה יש לו את המינימום הגבוה ביותר. כלומר - הוא יבחר באסטרטגיה שבה המינימום שלו יהיה עדיין גבוה.
  השחקן B יסתכל על העמודות ויבדוק היכן הוא יוכל לקבל את הערך המקסימלי הנמוך ביותר, ויבחר את האסטרטגיה הזו.

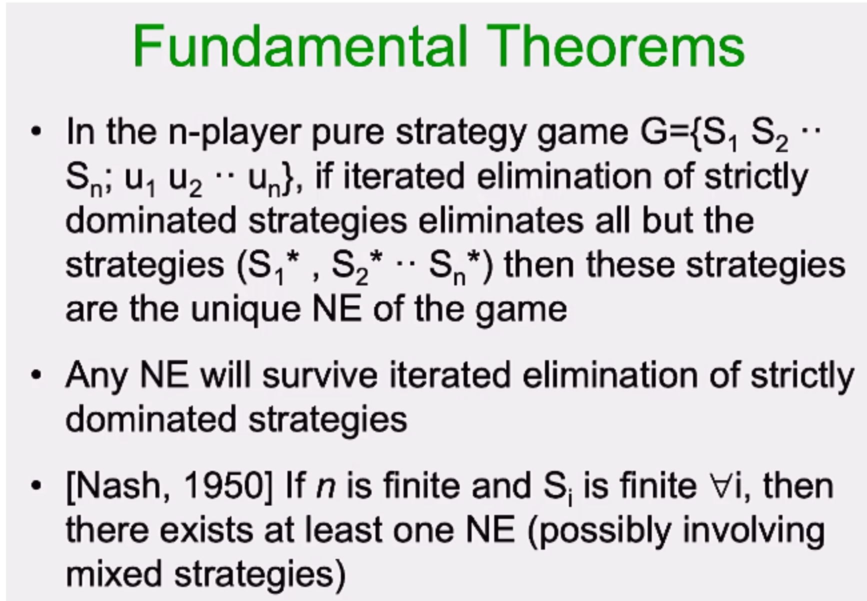
- ***mixed strategy*:** אנו נשתמש באסטרטגיה הזו בה ניתן ניקוד לכל בחירה, ולכן בחירה בדרך מסויימת תביא לנו מספר נקודות כפול ההסתברות p.

- **דילמת האסיר:** זהו לא משחק סכום 0 - שניהם יכולים להרוויח או להפסיד יחד. שני אסירים בכלא, אם שניהם מתוודים - יקבלו 3 שנים. אם אחד מפליל את השני - הראשון יוצא ללא כלום, והשני מקבל 20 שנה בכלא. אם אף אחד לא מודה - שניהם יקבלו שנה אחת בלבד. כעת עליהם להחליט באיזו אסטרטגיה לבחור. סתבר שאם שניהם יחשבו באופן שלמדנו הם יבחרו להתוודות שניהם ולקבל 3 שנים בכלא, בעוד האופציה הטובה בשביל שניהם היא לשתוק.

- ***Strict domimation*:** אסטרטגיה לפתרון בעיה שאינה משחק סכום 0 (כדוגמת דילמת האסיר). תחילה השחקן הראשון פוסל את האפשרויות הגרועות בשבילו ומוריד את השורות הללו מהמטריצה, אח"כ השחקן השני מבצע את אותן פעולות על העמודות וכן הלאה... עד שנגיע לתשובה מספקת. אך לפעמים אנו נתקלים במטריצות שאין לנו אף שורה או עמודה למחוק בהן, ולכן לא נוכל להשתמש בשיטה זו.

- ***nash equlibria* - נקודת שיווי משקל של נאש:** נקודת שיווי משקל במשחק היא צירוף של אסטרטגיות השחקנים, כך שלאף אחד מהשחקנים לא משתלם לשנות את האסטרטגיה שלו אם שאר השחקנים אינם משנים את האסטרטגיה שלהם.
  הערה: יכול להיות שיש יותר מנקודת שיווי משקל אחת, או שאין נקודת שיווי משקל כלל (בשקף הבא יש משפט האומר מתי כן יש נקודת משקל).

- **משפטים על נקודת שיווי משקל:**

- 

11.2 תרגול 11 - פרוייקט:

- לבחור רעיון שמעניין אותנו. משחק, בעיה בעולם האמייתי.
- נממש אלגוריתם אחד או יותר שבו ננסה לפתור את הבעיה או חלק מנה.
- נציג את הבעיה בסרטון. ונכתוב דוח.
- 11 ליולי צריך למצוא . צריכות להיות שתי דרכי פתרון לפחות - להשוות בין כמה היוריסטיקות או שני אלגוריתמי חיפוש, אנו צריכים להשוות בין כמה שיטות.
- להסביר למה הבעיה שלנו פותרת את הבעיה.
- למדוד את התוצאות בהתחלה ובסוף.
- דוח: מה הבעיה, באיזה אלגוריתמים השתמשנו, תוצאות ומסקנות מהתוצאות.
- סרטון: למה הבעיה מעניינת, למה הפתרון מתאים לבעיה, מה התוצאות.
- דד ללין להגשה 31.8
- בדוח צריך להביא רפרנסים להכל.
-

12 שבוע 12:

12.1 הרצאה 12:

12.2 תרגול 12:

13 שבוע 13:

13.1 הרצאה 13:

13.2 תרגול 13: