**פונקציות ממשק של AbstractState**

המצב האבסטרקטי:

* תומך ב:
  + query- האם קיים משתנה מסוים
  + set\_var\_to\_var- עדכון כאשר נעשית השמה מהצורה x=y, כאשר y אינו משתנה פשוט
  + set\_var\_to\_const- עדכון כאשר נעשית השמה מהצורה x=y, כאשר y הוא משתנה פשוט.
  + clone- החזרת מצב אבסטרקטי חדש וזהה לקיים
  + lub- ביצוע פעולת lub בין שני מצבים אבסטרקטים.
  + Add\_var\_

**תיעוד הגרף (בשביל העבודה העיונית)**

כל קשת בגרף מכילה:

* מי הצמתים שביניהם היא מחברת (אב ובן)
* התווית שלה
* רמת הידע שיש לנו עליה- איבר ב-lattice (שבפועל יהיה L\_MUST\_HAVE או L\_MAY\_HAVE, ואולי גם L\_MUST\_NOT\_HAVE)

כל צומת בגרף מכיל:

* אינדקס
* אינדקס של קבוע שהאובייקט שווה אליו
* רשימת שכנים (מופרד לאבות ולבנים). לכל תווית יכול להיות לכל היותר בן אחד, אבל הרבה הורים.
* רמת הידע על הצומת- איבר ב-lattice, שבפועל יהיה L\_TOP או L\_MUST\_HAVE

הגרף:

* הגרף הוא DAG (עד כדי לולאות עצמיות. כרגע אין טיפול בלולאות עצמיות)
* כל קודקוד הוא או TOP או שיש לו קבוע (צריך עוד לחשוב על זה קצת)

פונקציות עיקריות בגרף:

* השמה של משתנה לקבוע-
  + מסמנים שהקבוע של הצומת הוא ה-type של המשתנה.
  + מסמנים את הצומת כ-לא TOP (L\_MUST\_HAVE)
* הוספת צומת חדש-
  + דורש תווית ואב. אם הבן כבר קיים אצל האב, מוחק את הקשת מהאב לבן הקודם.
* הפיכה לאבא-
  + כמו הוספת צומת חדש, אבל בלי ליצור צומת חדש
* בדיקה האם קיים בן- (בהינתן אב ותווית)
  + בן יכול להיות קיים באחד מ-3 מצבים:
    - הקשת אליו קיימת
    - האב הוא TOP
    - באב מאוחסן קבוע שהבן יכול להיות attribute שלו
* מחיקה של צומת-
  + באמצעות ניתוק כל הקשתות (אבות ובנים) של הצומת
* Consolidate (garbage collector)-
  + מוחקים מהגרף צמתים שלא ניתן להגיע אליהם מהצומת הראשי.
  + מוחקים קבועים שלא בשימוש של אף צומת.
  + מעדכנים את האינדקסים שבשימוש כך שיהיו קומפקטים
* Lub
  + הספר מפורט יותר בהמשך

**תיעוד של המצב האבסטרקטי**

המצב האבסטרקטי תומך ב:

* פונקציית שאילתא למשתנה מסוים
  + מחזירה שגיאות אפשריות בגישה למשתנה
* השמה של משתנה לא פשוט-
  + עדכון כאשר נעשית השמה מהצורה x=y, כאשר y אינו משתנה פשוט
* השמה של משתנה פשוט-
  + עדכון כאשר נעשית השמה מהצורה x=y, כאשר y הוא משתנה פשוט
* clone-
  + החזרת מצב אבסטרקטי חדש וזהה לקיים
* lub-
  + ביצוע פעולת lub בין שני מצבים אבסטרקטים.
* הוספת משתנה שדבר אינו ידוע עליו, וסימונו כ-TOP-
  + הקשתות עד אליו יסומנו כ-L\_MUST\_HAVE (אם הן לא קיימות)
  + אבות של המשתנה יווצרו גם כן כ-TOP אם הם לא קיימים כבר
* מחיקת משתנה מהגרף-
  + עדיין לא קיים. האם צריך לעשות query ולתקן את הגרף?
  + גשדגדשגדש
* המרת ביטוי לצומת בגרף-
  + מטיילים על הגרף לפי התוויות, בהתאם לביטוי
    - מגיעים באופן רקורסיבי לאבא של הביטוי
    - אם הבן קיים בצורה לא מפורשת, מוסיפים אותו לגרף
    - אחרת, מחזירים שהצומת לא קיים
* Query-
  + עטיפה של הפונקציה האחרונה
  + זוכר את כל הדברים שיכלו להשתבש (קשתות לא קיימות, קשתות עם L\_MAY\_HAVE)
  + כל attribute שיכל להשתבש, נוסף לגרף. אופציונאלי. האם אנחנו רוצים את זה?
  + צריך להפוך קשתות L\_MAY\_HAVE ל-L\_MUST\_HAVE?

Lub בין מצבים אבסטרקטים

בשלב ראשון, נשנה את שמות הצמתים והקבועים, כך שלא יהיו שמות זהים (פרט לשורש שיישאר 0 בשני הגרפים).

נעבור על שני הגרפים ב-DFS במעבר ראשון, ונמצא צמתים עם אותו נתיב אליהם (וזה אומר שהם זהים). אם קיימות קשתות לבנים בגרף אחד ולא בשני, נבדוק האם הקשתות יכולות להתווסף גם בגרף השני (אם האב הוא TOP או מכיל קבוע כלשהו), ואם כן נוסיף את הקשתות ואת הבנים הרלוונטים (ונמשיך לטייל על הבנים ב-DFS).

נעבור על שני הגרפים מעבר שני (ב-DFS), ונחלק את הצמתים והקשתות ש:

* נמצאים רק בגרף הראשון
* נמצאים רק בגרף השני
* נמצאים בשני הגרפים יחד

טיפול ב:

* קשתות משותפות:
  + עושים lub בין ה-knowledge שלהן.
* צמתים משותפים:
  + איחוד של כל הקבועים שהם אולי שווים אליהם
  + Lub על ה-knowledge שלהם (אם אחד הוא TOP, אז כך יהיה גם האיחוד)
* צמתים וקשתות לא משותפים:
  + כל הקשתות הופכות להיות L\_MAY\_HAVE (כולל הקשת שמחברת צמתים "ראשיים" לצומת -1)
  + צמתים **לא** נהפכים להיות TOP.
  + צריך לשים לב מה קורה כאשר שני צמתי הקצה משותפים, אבל הקשת לא.