TLV-BASIC - כתיבת חוק הכרה

כיום, הצורה שמאפשרת נוח לבטא אלגוריתמים

של המערכת.

כמו שאלות במערכת האינטראקטיבית

קיימים כלים שיכולים שונים כולם

לבטא קריאה גלובלית.

המערכת שלי כמו הם ללמד איך [illegible] לייצג

את האלגוריתם הזה של מי שנותן

לפתוח בה במערכת האינטראקטיבית (תרגיל 3)

* הערה

כמו שאני עובד עם שני ~~[illegible]~~ שפה ה-VMS

ה-VLT כמו שמשלבים במערכת ה-TLV

אז כן קיימים לנו שני סוגי משתנים! משתנים מערכת

שאלו המשתנים שהם חלק מהמערכת שלנו כיוון

לבנון (VMS) הם מוגדרים [illegible] ROBOT

שמוגדרים על המערכת

22

ישנם שני סוגי משתנים כלל משתנים

גלובליים כמו מספר ___ כמו ה-TLV

כמו ROB00 או משתנה: כלוי הפעלה

כמו ה- Let

את משתנה לא צריך להצהיר מראש

ישר להשתמש בהם (כמו ב-BASIC)

אפילו את מערכים לא צריך להצהיר לפני

(23)

```
Let var := expr;
```

```
Proc proc-name(parm1,...,parmn);
    S
End
```

```
While (expr)
    S
End
```

```
IF (expr)
    S1
Else
    S2
End
```

```
Call proc-name(parm1,...,parmn);
```

טיפוסים ב-TLV

- TLV משתמש אך ורק ב-Bdd's

* כאשר מילה עם-Bdd שמשתמש ב-TLV
מתארים עם ערכים אחרים מ-0 ו-1 אז
יכול להיות שזה [F] (נושא לדיון)

- בעיות השימוש ב Bdd נובעות מכך שאם
יכולים להיות הרבה משתנים כמו במצבי תוכנה
ערכים שלמים, קבוצות מספרים וכו'
השאלה היא רק כמה קשה נראה להיות
של ה Bdd

- כמה משתנים הניתנים של הTLV מאפשר
בסיסי: פורמט: atom
או
atom[index]
הגדרה: מערכים הם
ישנה אין דרך לשלוט על האורך אך
בזמן ריצה צריך לק שכזה ...!

(25)

משתנים מוצהרים מראש

בגלל שאין מערך ב-VMS.

מערך ב-TLV יוצר משתנים דינמיים

מקושר של מערך כו (פרק 9 עמ' 14-23)

פרק 4

למשל (תרגיל):

[illegible]

* total[i] = t[i]^d[i]^pres[i]_

* i[i]_

ה משתנה שמוגדר בתוכנית VMS. נניח XX

[illegible] וקיימים גם מערך שלו

אז XX[i] יהיה ... XX של מערך

(אפילו אם XX במערך ... XX[i]

הוא מערך

---

bi_ - ה משתנה [illegible] שלו כמה מספרים ...

var_ - ה משתנה ...

(62)

ביטויים

לוגיים (שקף 7 מ"ט) **שקף 5**

אופרטורים לוגיים רגילים. מחזירים ערך 0/1

מספריים

...

קבוצה

ביטוי כזה יוצר קבוצה אפשרית **שקף 6**

{val1, val2, ..., valn}

אופרטורים על קבוצה (שקף המשך של מ"ט 16) **שקף 8**

| expr1 in expr2 |

מחזיר 0/1 אם expr1 [illegible] שייך/מוכל בקבוצה expr2

| expr1 notin expr2 |

שלילה הקודמת

| expr1 union expr2 |

יוצר קבוצה המספר של האיחוד

(אם יש את האלמנטים האלו מספר פעמים עדיין מופיע פעם אחת)

(28)

שקף 6

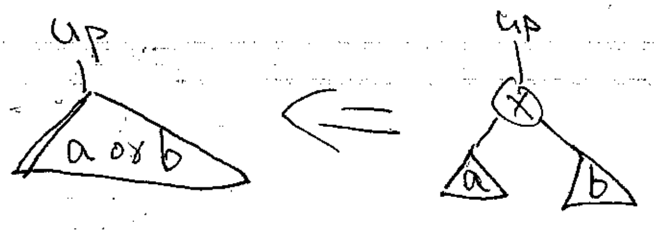
פעולה נוספת על קבוצה

expr1 forsome expr2 ← חיבור המצב

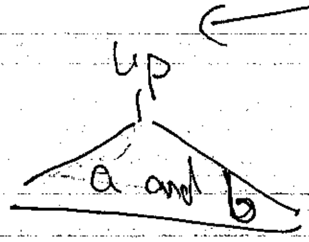
מחזיר T אם מערכי expr2 רק זה מתקיים

הערך שלהם

יש $x \in expr2$ כך ש



ובאותו אופן forall



שקף 17 שאר הפעולות על קבוצות — שקף הבא

דוגמה בשאלה קבוצה (לא נכנס לחומר מבחן)

[illegible]

sat(expr)

מחזיר שלם עם מסלול אחד לכמה מצבים (דוגמה כמו ספונטני)

fsat(expr)

לכן תשובה

כל הוגן 3 פה נמצא מסלול רבים

fsat(expr)

fsat(expr, set_expr)

מחזיר פונ' אחד יחיד כל כל המשת

(מה שלא היה חשוב - מה שיכול להיות בהמשך

→ ה סופי)

אופרטורים אחרים

תשובה

```
case
    expr_a1 : expr_b1 ;
    . . . . . . .
    expr_an : expr_bn ;
esac
```

מושג הגדרה דומה לחישוב

ב- SMV.

מחשב

```
next(expr)
prime(expr)
```

מחזיר את קבוצת המצבים הבאים האפשריים

```
unprime(expr)
```

מחזיר את קבוצת המצבים הקודמים האפשריים

לא חשוב

```
expr1 && expr2
```

מקביל ל- sat(expr1 & expr2)

```
size(expr)
```

מחזיר את גודל ה-BDD של הביטוי expr

```
exist(term)
```

מחזיר 1 אם מופיע term בביטוי כלשהו

(31) succ(trans, state)

$$\text{unprime}\left(\exists V : \text{trans}(V, V') \wedge \text{state}(V)\right)$$

pred(trans, state)

$$\exists V' : \text{trans}(V, V') \wedge \text{state}(V')$$

assign (dvar, pvar, val)

Value (expr)

Value (dynamic-var, system-var)

(33)

פקודות בסיסיות

(פקודה זו עושה [illegible] [illegible])

```
Let variable := expression;
```

יוצר ROBDD ל-expression ושומר אותו בתוך המשתנה variable

variable יכול להיות atom או atom[index]

אין צורך להכריז עליהם לפני

למשתנים שמופיעים ב-VMS לא צריך להכריז [illegible] [illegible]

```
Print expr, expr, ... ;
```

* לכל expr הוא מדפיס [illegible] [illegible] שלו שיכולה להיות (אם expr הוא string אז הוא מודפס)

```
Load "file-name";
```

סיומת קובץ .tlv

34

Exit n;
Quit;

(35)

פונקציה — שקפים 3-2

יש שני סוגים שונים: פונקציה (Func, To)

ופרוצדורה (Proc) ← כמו אלה name space שונה

לכל אחד namespace שונה

הצהרת פונ' נעשה ע"י:

```
To   proc-name  [parm1, ... parmn];   ← אופציונלי
        ......
End
```

או ע"י

```
P גדול → Proc   procname  ([parm1, ... parmn]);   ← אופציונלי
        ......
End
```

```
Func   func-nam  ([parm1, .... parmn]);   ← אופציונלי
     - . - - -
     Return....
End
```

func היא מחזירה ערך Return

קריאה לפרוצדורה / פונקציה מבוצעת ע"י

Run proc-name [expr1, ... exprn];
Call proc-name ([expr1, ... exprn]);
proc-name [expr1, ... exprn]

החזרת הערך מפונ' חייבת להיות ע"י

Return [expr];

נשים לב שיש 2 ביטויי השמה

הראשון (Let) מקומי כלומר אם

נגדיר בתוך פונ' Let a:=8

אז כשנחזור מהפונ' נקבל ל-a

(כלומר גם כן יכולה להחזיר רק ערך)

אם מגדירים בתוך פונ' נקבל

השני ביטוי השמה

Local a:=8

אז a יהי צמוד לסקופ

(* הערה - למחר פרמטר a אין צורך להשתמש
ב-Local אלא להמשיך להשתמש ב-Let

שקף 24

(37)

הערה:

* נשים לב! שבשורה שלפני ט"ו מרכיב
את הקריאה אליו מבצעים ע"י ~~[illegible]~~

(proc-name ⎵)

↑ רווח חובה

(אחרת ה-interpreter יבלבל את האופרטור "()")

* ~~[illegible]~~

ניתן להעביר את הפרמטרים ע"י call-by-name ע"י הוספת "&" לפני המשתנה

(הקריאה רגילה כמו ע"י call-by-value)

(אפשר לראות בשקפים עמ' 23-22)

(38)

פקודות נוספות

While (expr) statement End

If (expr) statement$_1$ [Else-st$_2$] End

Break; → יוציא מהלולאה

Continue; → חוזר לתחילת הלולאה

שקף 7 — שקף באמצע דף 23-24

הערה: !

If (k=0)

End

מחזיר k - 1 כמו bold של if השמאלי

ואז חוזר ~~[illegible]~~

true כי ב-k יש מספר גדול

39

לכן צריך לשאול על ה-if הזה
עבור

If (k)

הפסק ...

Else

... = מה שרוצים

End

זה אומר
שיש ל- k
פשוט זה מה
שהתכוונו ב- k=0

ביטוי ParseTree (quote)

לא מעניין!

פרוק של סלים מאקרו

לא 6 - רק מעניין!

proc;
funcs; — המאפיינים עד כה

stats; — סטטוסים

ספר המשתמש ROBOT

לו מרכ'ן

- reorder; – בחירת סדר חדש
- sorder "filename" – שמור סדר מושלם
- lorder "filname" – טען סדר מהקובץ

. . . .

פקודות פלט

לו מכ"ן

– – – –