(1)

TLV (Temporal Logic Verifier)

(2)

(1)

(2)

(3)

(4)

(5)

*

(3)

(1) tlvsun

rules.tar.Z

~[user]/TLV/

(2) zcat rules.tar.Z | tar xvf -

(3) ~[user]/.tcshrc

"... ## LOAD CONF ##..."

setenv TLV_PATH "~[user]/TLV"

alias tlv "~[user]/TLV/tlvsun"

(4) Logoff , Logon

(5)

% tlv [option] [file.tlv] file.smv

your wish is my command.....

(4)

option

- לקרוא ב-manual : כל דבר עקרוני [illegible]

[illegible] את המופע [illegible]

(המופע ה- /לפעמים/ יותר מעניין בקוד)

או יותר [illegible] מעניין בקוד

או שונה ב-gui

או [illegible] ב-C (וכו'...)

אם [illegible] [illegible] ב- VLT ?

* <CTRL-D> [illegible]

- VLT [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] !!)

- [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

"END" או "." [illegible] [illegible]

- tlv [illegible] ב-gnu readline [illegible] [illegible] [illegible]

[illegible] delete וכו'

(5)

SMV

נועד לתאר את המערכת ← זה מערכת שניתן לתאר ע"י SMV ניתן לייצגה

כמות תאר מערכת — FKS (as in Finite) ה- SMV

יתאר לי ROBDD שמייצג של המערכת (אין משמעות רק בבוליאנים)

המערכת יכולה להיות לתאר חומרה, תוכנה ...

לתאר מערכות סינכרוניות, אסינכרוניות ....

2 — SMV man

● דוגמא! המשרה my example.smv (כמה נבוא [illegible] יש תוכניות.....)

ה- button של משנה [illegible]

machine שלו [illegible] את המצב מתוך המערכת (עם שלושה ערכים)

כמה משתנים בוליאנים שמייצגים את (קידוד)

לכל ערך כזה של machine ניתן string (busy, ready...)

(כי [illegible] [illegible] [illegible] [illegible])

- - - - - -

● המצב ההתחלתי נקבע ע"י ה-init(..) (machine שלו בחירה לא דטרמיניסטית - [illegible])

מבנים [illegible]: המצב הבא נקבע ע"י [illegible] ה- next(..)

הערך שיוחזר יהיה לפי ... : ה- case

שאומר שתחזיר הערך הראשון מבין אלו שבו תנאי מתקיים

(שים לב שניתן להחזיר ערך לא דטרמיניסטי!)

* טוב ל- : - מקומות של [illegible] - מגדלים אבסטרקציה

- אי אפשר בדיוק לקבוע ערך של משתנה

(6)

כשיש לך על- button יש לו לשנות

שמה על המקום בו הם text מה כתוב

ואחר לבחור (או ימין עם לחיצה על 2 הכפתורים)

(7)

3 SMV man

תרגול 2 - ספר

דוגמה ___ שימוש חוזר ב modules

תרגול מונה בינרי

* ראשית ___ נשים לב של- MODULE main יש משתנה מיוחד

שהוא בלי ארגומנטים ( )

* סדר ה MODULES . לא משנה ___

* נשים לב איך ___ instance של MODULE

- יוצרים counter_cell עם ערך אחד (carry-in)

[illegible] מצב של "1" אז

ע"י a.b של רכיב b בשדה a

* DEFINE עוזר כמו הגדרה על carry_out

אותה תוצאה היה לא מעט יעילה

```
VAR
  carry-out : boolean;
ASSIGN
  carry_out := value & carry-in;
```

- עוד לא ___ מחושב כל לא ידוע.

- היתרון ב Define הוא לא צריך [illegible]

לא יכול ___ להיות אי-דטרמיניזם!

(8)

<u>פתרון:</u>

1) הפעלת ה-VMS מתוכנה במקביל וביחד
(מה שאומר שצריך 2 תוכנות ה-VMS בין לינוקס), בין הסבר כל סוג המעבד
- interleaved מה של הבקשות מבוצעים
2) קשר (פרוטוקול מיוחד)

עד לכאן שאלה 1
כמה מילים על...

אחר להכניס פיזית 2 Junk

SMV - סינטקס וסמנטיקה

* atom - {A-Z, a-z, 0-9, _, -}

* מספר ...

* number - מספר

ביטויים

* על - 10 ב- SMV-man

atom, number, id, !, &, |, <->, ->, =, !=, <, >, <=, >=, +, -, *, /, mod, next([id])

case_expr, set_expr

הערה

סדר חישוב:

*, /

+, -

mod

=, !=, <, >, <=, >=

!

&

|

->, <->

הסדר ... מימין לשמאל

חוץ מ- -> ש...

ביטוי case

```
Case
    expr_a1 : expr_b1;
    expr_a2 : expr_b2;
        :
    expr_an : expr_bn;
esac
```

מחזיר את הערך הראשון מצד ימין רק שבו שמאל מתקיים

אם אף אחד לא מתקיים אז מוחזר ערך מספרי

(יש להיזהר עם זה ← צריך לכסות מקרה כללי default ← else כמו "1 : __")

כי יש צורך להחזיר ב-expr_ai ערך שהוא שונה מ - 0 או 1

ביטויי קבוצה

{val1, ..., valn},

expr1 in expr2,

expr1 union expr2

הפעולה הראשונה היא לבדוק את המשמעות

in (הכלה) האם expr1 בתוך expr2 (מחזיר 0 או 1)

ה-union מחזיר את הקבוצה של האיחוד

משתני מצב

VAR

```
atom1 : type1;
atom2 : type2;
...
```

ה-type יכול להיות ← MODULE, scalar, Boolean

או מערך של הדברים הללו. הסינטקס של type הוא:

(1) { val1, ..., valn } → atom numeric

(2) array [expr1]..[expr2]; [of [type]]

1..5 ← חייב להיות מספרים

(3) מופע מודול ← atom [(expr1, ..., exprn)] ← אופציונלי פר' במודול

(4) מופע מודול אסינכרוני ← process [(expr1, ..., exprn)] (atom)

(5) boolean

יכול לקבל ערכים מספריים 0 או 1

פסוקי השמה

ASSIGN

$dest_1$:= $expr_1$;

$dest_2$:= $expr_2$;

...

המיקום של dest יכול להיות

(1) atom — ערך הנוכחי

(2) init([atom]) — ערך התחלתי

(3) next([atom]) — ערך הבא

אם $expr_1$ שווה ערך מספרי או קבוע

זה המצב של הערך

אם expr הוא קבוצה של ערכים זה

הבחירה שבה הוא יכול אי-דטרמיניסטי

! חשוב: אם הקבוצה היא יכול גם קבוצה

כל הערכים האפשריים ל-dest זה מה שיהיה!

(! אסור להשתמש בפיסוק)

~~אסור שבפסוק יהיה יותר של מודל אחד פעם~~

~~אחד ולא של משתנה אחר לבד או מחוץ~~

כמו: אסור

next(x) := !y

next(y) := !x

סוגי שאסור לבצע במשא

1. ל-next יש לבצע הצבות ניתנים יותר משמה אחת בפקודה (בודד)
2. לבצע הצבה ניתן יותר מהשמה אחת
3. לבצע הצבה והתחלה ניתנים השמה
4. ערך הצבה והתחלה ניתנים השמה

הכרזת מעברים — למשתנה כזה יש דינמי

של ערך נוכחי / ערך הבא

השמה של ה next

TRANS expr

! (הביטוי חייב להחזיר 0 או 1)

איחוד של כל תנאי המעבר בביטוי אחד:

$$expr = (\text{תנאי מעבר}) \wedge (\text{תנאי מעבר}) \wedge \ldots \wedge (\text{תנאי מעבר})$$

[illegible]

הכרזת איתחול — למשתנה שאין יחס אונרי של

ערכים שונים במצב יכולה להתחיל

INIT expr

! הביטוי חייב להחזיר 0 או 1

! אסור לו להכיל next

איחוד של כל תנאי ההתחלה של המערכת

$$expr = (\text{ערך התחלתי}) \wedge \ldots \wedge (\text{ערך התחלתי})$$

DEFINE

```
atom1 := expr1;
atom2 := expr2;
...
```

INVAR

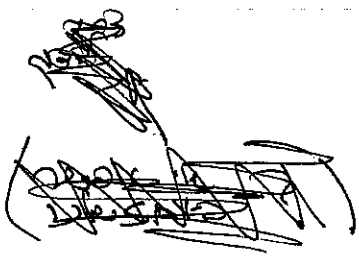
```
INVAR expr
```

MODULE atom [(atom1, ...atomn)] ——— MODULE

decls | decl1
decl2
....
decln

Identifiers

atom
id.atom
id[exp]



main MODULE

תבנית מערכת :

module1
module2
. . . .
modulen

אחד מהמודולים ← חייב להיות main

תהליכים (processes)

תהליך הוא מודול מופעל ב- process ;

- תהליכים לא מסונכרנים interleaved

כלומר ב כל פעם מתבצע צעד בודד של תהליך
לא ידוע מראש מי יבצע את ה הצעד שלו בתורו

(לא ב-SMV שלנו יש לכל תהליך כמה יש
משתנה running שהוא 1 אם התהליך רץ
כרגע)

-

(אפשר להגדיר בשפה תהליכים עם כלן)

מערכות (systems) — תוספת של ה-SMV שלנו

בפועל מגדיר תהליך רק שבו המבנה
כאן שבו מערכת שהם תהליכים והמיוחד
של ה-VLT סינכרון מהותי יותר מה שהם

בירור בסוף סמסטר

(טלפ)

הכרזת משתנים פנימיים

OWNED

הסינטקס דומה בדיוק ל- ASSIGN

הסמנטיקה היא כמעט זהה ההבדל הוא

שאלו משתנים שלא ניתן לגשת מחוץ לתהליך

(מעין private)

JUSTICE

expr

COMPASSION

expr

```
0: while forever {
1:     noncritical
2:     wait while y=0
3:     critical
4:     y=0
```

(19)

~~הרכב~~ מרכיב
~~הרכב~~ מרכב

COMPOSED expr

תוסף נוסף שהוסיף לו ה-tlv הוא

האפשרות להגדיר בדרך שכזו מרכיבים

של המרכב

! אם הרכב זה קיים במודול אז

לא ניתן להגדיר לו TRANS, ASSIGN ו-OWNED

למרכב המודול הוא מודול מיוחד

שמיועד רק להרכיב את המרכב

! גם כך ל-VAR יש משמעות שונה

בכך שהיא מתארת משתנה שהוא אך ורק

לא הם מורכב ב COMPOSED

- להגדרה ה COMPOSED יש מוספת עוד שני

אופרטורים || - הרכבה סינכרונית

||| - הרכבה אסינכרונית

(פירוט עמ' 38 ב tlv manual