~~0. מעבר על הקוד הקיים~~

~~1. תשתית (שקפים 112-114, פרק 6) - "לתמוך בכלום בצורה מושלמת"~~

~~a. כל הקבצים מהעבודות מתחברים זה לזה ומדפיסים שגיאה not\_implement~~

~~b. קוראים את הsrc file ומשרשרים אליו את הקבצי ספריה שלנו בscheme (וכמובן מוסיפים את קוד הscheme שקיבלנו (stdlib.scm))~~

~~c. כותבים קוד לבניית טבלת קבועים (שקף 36)~~

~~d. כותבים קוד לבניית טבלת משתנים חופשיים (שקף 63)~~

~~e. בניית השלד (Prologue + Epilogue + nasm + gcc...)~~

~~e. כותבים סקריפט טסט לבדיקה כמו שהם יעשו (סעיף 5) – מה שהבחור שלח בוואטסאפ~~

**(סוף יום 1)**

~~2. Consts & test~~

~~3. Seq & test~~

~~4. If & test~~

~~5. And & test~~

~~6. Or & test~~

~~7. Define & set - free variables & test~~

**(יום 3)**

8. ~~מבטלים את ApplicTP'~~ ו:

~~a. LambdaSimple~~

* ~~calc |Env| - not necessarily in assem~~
* ~~allocate ExtEnv – size of 1 + |Env|~~
* ~~copy pointers of minor vectors from Env (through stack) to ExtEnv with offset of 1~~
* ~~Get the number of params from stack~~
* ~~Allocate ExtEnv[0] to point to a vector in the size of Params (as stored in the ArgsNum on stack)~~
* ~~copy parameters from the stack to that vector~~
* ~~generate the Lcode label~~

~~b. Parameters / get~~

~~c. Paramenets / set~~

~~d. Bound vars / get~~

~~e. Bound vars / set~~

~~f. Box / get~~

~~g. Box / set~~

**(יום 5)**

~~8ב. Applic~~

~~Test a lot - כאן יהיו רוב הבאגים, צריך להגיע לזה מהר ולהקדיש לזה זמן~~

~~למבדות~~

rec1 – הפונקציה הרקורסיבית func דורשת בוקס למיטב הבנתי.

בזמן יצירת הclosure של ה lambda הפנימית (זו שנמצאת בrhs של הset! אחרי שמרחיבים את הLetrec לlambda simple, הקוד שאנחנו מייצרים נמצא בהערה בתוך הטסט). אנחנו מרחיבים את הסביבה.

בזמן ההרחבה של הסביבה אנחנו מעתיקים לסביבה את PVAR(0) לheap למקום ה[0][0] שכרגע הוא מצביע לסימבול whatever בטבלת הקבועים. אחרי שאנחנו מסיימים לייצר את הclosure (שסוחב איתו את הסביבה) אנחנו דורסים עם set! את המשתנה PVAR(0) על הstack עם הכתובת לclosure שהרגע ייצרנו. בסביבה של הclosure בheap עדיין יש מצביע לסימבול של whatever (כפי שהיה בstack). לכן כשאנחנו באים להפעיל את func בתוך הif, אנחנו פונים ל VarBound(func,0,0) שמכיל מצביע לסימבול whatever. כשאנחנו מנסים לקרוא לזה, יש segfault (אם משאירים את הבדיקה של הclosure type אז יקפוץ int3) .

אם עושים בוקס (הכרחתי בsemantic-analyser לעשות בוקס להכל) אז הקוד עובד נפלא.

[צריך לשים לב שאם עושים include ל stdlib.scm אז יש segfault בלי שום קשר כי גם שם יש פונקציות רקורסיביות]

**(יום 6)**

9. ניתן לבדוק את (תלוי רק במשתנים חופשיים ואפליקציות):

all this problem are because the variadic versions run over these address with void (lambda opt is missing for map)

+ (binary version), -> test arithmetic2 does some crazy shit

\* (binary version), -> test arithmetic3, same shit

/ (binary version), -> test arithmetic4, same shit

< (binary version), -> test arithmetic5, same shit

= (binary version), -> test arithmetic6, same shit

~~boolean?,~~

~~char->integer,~~

~~char?,~~

~~denominator,~~

~~eq?,~~

~~exact->inexact,~~

~~flonum?,~~

~~gcd (binary version),~~

integer->char

ASCII (<127) integers fails just because representation differences ( #\x2 <-> #\x02 )

Non-ASCII (>127) and "DEL" (=127) integers fail. Should ask in the forum.

**TEST NAME: integer\_to\_char2.scm**

make-string (binary version)

Unary varsion fails. But in this case chezscheme says the characters

contained in the string are unspecified.

So I guess we're fine. And, it might magically work after allowing stdlib functions

~~null?~~

~~pair?~~

~~procedure?~~

~~rational?~~ Fails but we're OK.

From the assignment page:

"Applying rational? to a floating-point value almost always returns #t in Chez,

while our implementation returns #f for all floating point numbers."

~~string-length~~

~~string-ref~~

~~string-set!~~

~~string?~~

~~symbol?~~

symbol->string

Doesn't work with capital letters (our compiler converts them to lowercase)

**בפורום הם כותבים מדי פעם שהם מגלים טעויות בקבצים שהם מעלים, צריך לוודא שאנחנו עושים כל הזמן update כדי שלא נסתבך סתם.**

~~9ב. ניתן לבדוק את (תלוי ב lambda simple):~~

~~a. not~~

**(יום 7)**

~~9ג. מתחילים לממש פרימיטיבים~~

~~a. מממשים באסמבלי (סעיף 4.1)~~

~~car, cdr, cons, set-car!, set-cdr!~~

~~ובודקים~~

b. מממשים בscheme את:

~~fold-left (non-variadic!), fold-right (non-variadic!), and cons\* (variadic)~~

**ובודקים**

**(יום 8)**

9ד. ניתן לבדוק את:

integer?, number?, zero?

9ה. ניתן לבדוק את:

a. \* (variadic), + (variadic), - (variadic), / (variadic), < (variadic), = (variadic), > (variadic), append (variadic), equal?, gcd (variadic), length, make-string, map (variadic), string->list.

~~9ו. לא לשכוח לבדוק pset~~

לא לשכוח שבchez אין תמיכה בpset

**(יום 9)**

~~10. מחזירים את ApplicTP'~~

~~a. מוסיפים תמיכה בTP~~

~~b. Test a lot - גם כאן יהיו הרבה באגים~~

~~c. מימוש באסמבלי של apply (variadic)~~

**(יום 11)**

~~11. LambdaOpt & test~~

12. מוסיפים את שאר הפרימיטיביים (תלויים בlambda opt)

List

**(יום 13)**

**סיכום והמשך עבודה**

* בעיית הבוקסינג בטסט rec1 – לבדוק האם אנחנו אמורים לעשות בוקסינג לפי עבודה 3 (הסבר כתבנו למעלה באדום)
* טסטים שצריך לבדוק למה נכשלים:
  + foldright.scm
  + is\_rational.scm
  + whatsapp6.scm
  + whatsapp7.scm
  + whatsapp9.scm
* טסטים שצריך לבדוק כי הם חשודים בגלל בעיות heap:
  + tp7.scm
  + whatsapp2.scm
  + whatsapp5.scm
* טסטים שאפשר להריץ רק ידנית (אמור לא להיפסק):
  + tp9.scm
    - 11.1 בשעה 10:00 – רץ למשך שעה בלי להפסיק.
    - אפשר להריץ את זה ובמקביל להריץ גם גרסה ללא אופטימיזצית tp ולראות אם השניה קורסת לפני הראשונה.
* טסטים שלכולם יש בהן בעיות (כנראה בגלל הסגל):
  + arithmetic2.scm
  + arithmetic3.scm
  + arithmetic4.scm
  + gcd.scm
* טסטים שנכשלים בגלל הבדלים בהדפסה:
  + applic15.scm
  + closure1.scm
  + const3.scm
  + define2.scm
  + exact-to-inexact.scm
  + integer\_to\_char1.scm
  + lambda\_opt\_closure.scm
  + make\_string1.scm
  + try.scm
  + whatsapp1.scm
  + whatsapp1a.scm
  + whatsapp1b.scm
* טסטים שנכשלים בגלל התנהגות שונה של chez:
  + eq-consts.scm
  + eq-false.scm
  + pset.scm
  + whatsapp8.scm

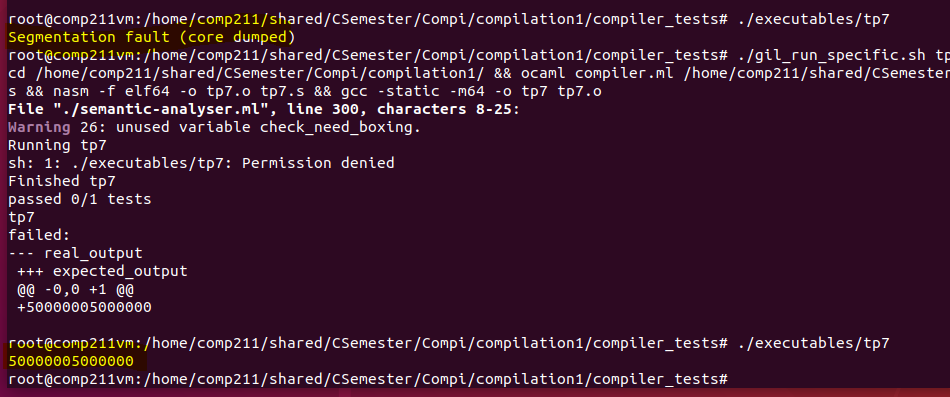
בנוסף צריך להשלים את הטסטים:

* list
* fold-left (non-variadic!)
* fold-right (non-variadic!)
* cons\* (variadic)
* integer?
* number?
* zero?
* \* (variadic)
* + (variadic)
* - (variadic)
* / (variadic)
* < (variadic)
* = (variadic)
* > (variadic)
* append (variadic)
* equal?
* gcd (variadic)
* length
* ~~make-string~~
* map (variadic)
* string->list.

בנוסף:

* לוודא שעושים git pull לעדכונים כשמבינים שהם בסדר.

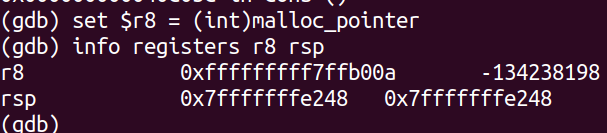
דיבוג tp7

* 

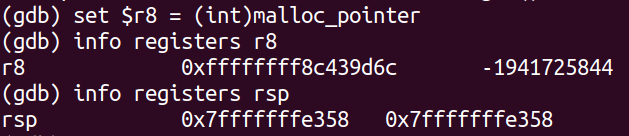
ריצה ראשונה – עם stdlib

ריצה שניה – ללא stdlib

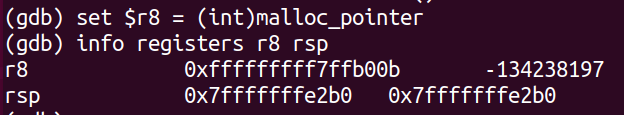
הריצה הזו עם stdlib ועם 6GB מוקצה ל heap:



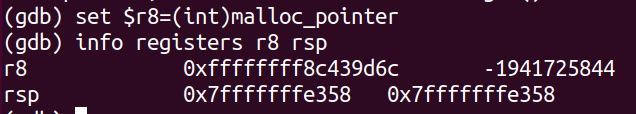
הריצה הזו ללא stdlib ועם 6GB מוקצה לheap:



הריצה הזו עם stdlib ועם 2GB מוקצה לheap:



הריצה הזו ללא stdlib ועם 2GB מוקצה לheap:

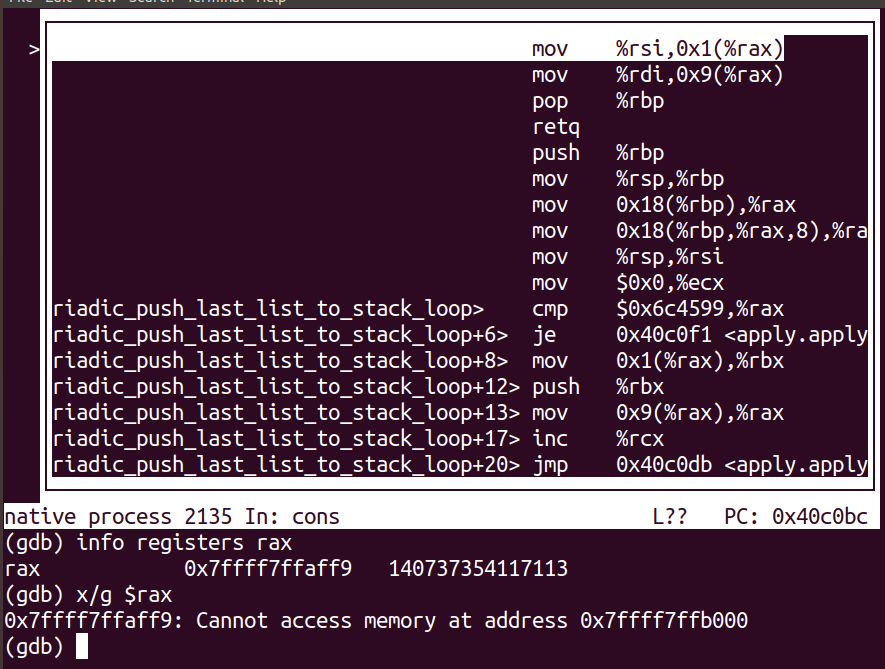


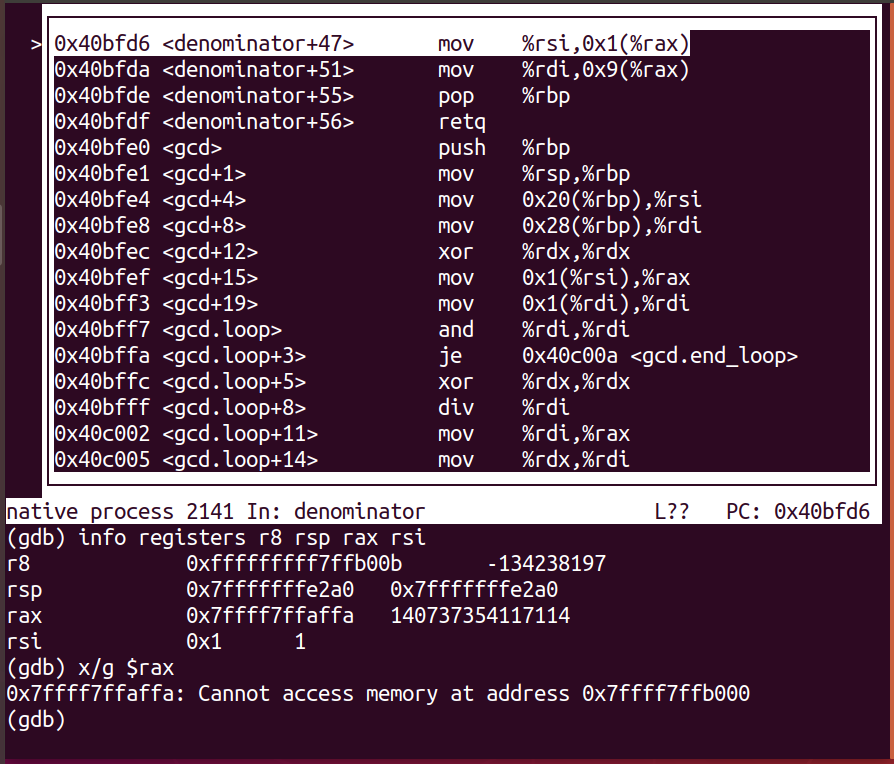
מסקנות:

1. נראה שאין הבדל בצריכת הheap של stdlib (לא משנה כמה מקום מקצים)
2. נראה שאין הבדל בין צריכת המחסנית כשלא משתמשים בstdlib
3. נראה שיש הבדל במחסנית כשמתמשים בstdlib אבל הוא הבדל של 160~ בתים, לא אמור לגרום לoverflow ולא הסיבה לקריסה.

איפה נתקעים כשמריצים את tp7 with stdlib 6gb:

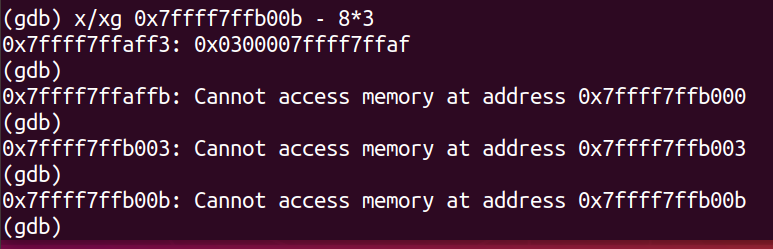
נתקעים בcons, כשמנסים לקרוא את PVAR(0) ולשים בתוך rsi.



בעיה דומה מקבלים כשמריצים את tp7 with stdlib 2 gb:

לא ברור למה הכתובת חורגות ממה שיש בheap מאחר שהmalloc pointer מצביע לערך שנמצא אחריהן, כלומר אמור להיות בכתובות האלה משהו.

אלו הכתובות האחרונות בheap:



מבחינת הstack\_trace, הוא נתקע שם בתוך fold\_left, map, map\_one

האמת, שאני ממש לא מבין מה קורה כאן