

# Forum:[Hack Forums](#)

Thread: [\[In progress\] The Twi Programming Language - A GPU Powered Programming Language](#)

**BlackCap** - July 28, 2014, 9:50 am

This thread is divided into 4 parts: about, tutorial, why the language is awesome, and finally some questions. Each part will have a big header in case you want to scroll past a part.

At the bottom of the tutorial part there is a link to a short but more or less complete language reference, but I'd recommend reading the "Why it's awesome" part first for inspiration.

If you're just curious about the syntax, here's an example:

<https://dl.dropboxusercontent.com/u/69377299/TwiSite/std.html>

====About====

I have been lazily working on and off on this language for about a year and a half. The goal is to create a high level language that is as fast as possible.

I don't really have a working compiler yet, but I do have interpreters in various languages, and a partly functional Twi-to-Javascript compiler.

====Tutorial====

Lambda (Taken from [this page](#), skip if you know Lambda calculus)

One day, a little boy named Church grew curious and wanted a language for all the school algebra and arithmetic computations he's been learning:

$1 + 3$

"Okay, I'll just compute this as 4". Encouraged by his little victory, he moved on to a slightly more complicated problem

$5 + x + 1$

he quickly realized that he couldn't do any meaningful "computations" on this little guy because he had no idea what that x stood for.

"Hmm, maybe if I ordered a guard to stand in front of my expression as if it was a tiny room and ask for what x stood for before letting anyone in!" and so little Church jotted down

Ask-for x.  $5 + x + 1$

"and once the missing x is accounted for, the guard no longer needs to guard the little room; it's all so perfect!"

but he quickly grew tired of writing down "Ask-for" all over the place, so he decided to name his guard ? because it looked like a little person to Church.

? x.  $5 + x + 1$

Now, what should ? do with that x once he receives it? It might not be at all obvious to you and me, but it was all clear as daylight to little Church. "Replace all the x's with guarded by ? with what he got!" he would yell; and so if I gave ? a little present containing " $1 + 2$ ", ? would go inside, and scrub

out all of the x's and replace them with 1 + 2, adding in parenthesis whenever he starts feeling claustrophobic

$5 + (1 + 2) + 1$

of course, we can now evaluate this into just good ole 9; and so little Church, after yet another conquest, once again became content.

Now you might protest: how can we tell Sir ? (after his heroic deeds today, I'm sure you'll agree that he deserved his recent knighthood) what x is? He'll listen to you, but I don't know how to talk to him!

"But it's so easy" he says with a wink and a smile "all you need to do is put it next to ?'s little room, and he'll come out and fetch it all by himself", and so if we set a 3 next to ?'s little room  
(? x. x + x) 3

then ? will come out, fetch the three, scrub the x's, and leave you with a present

$3 + 3$

of course, ? doesn't care if it's just a small number outside his doors or a another guarded room (or even another guard fetching his own x), it'll bring it into the room irregardless, so if you left  
(? x. x + x) (? x.x)

you should expect to find, sitting out your doorstep the very next day:

$(? x.x) + (? x.x)$

and with such a formidable language of computations at his disposal, little Church went to bed.

Let's review: little Church constructed a language consisting of numbers, variables, and addition. In addition, he created these guards called ? guarding some variable x that blocks computations until we give ? an x, and to do so, we just place the value to give to these guards right next to them (well, to their right).

Twi

Now, its kinda mean to make the guard walk all the way to the right to get whatever we left him, so lets place everything to the left instead:

$5 (\backslash x. 2 \ x) = (2 \ 5) = 10$

This was all nice and dandy for a while, but now the mathematical operators felt discriminated - they actually have invisible guards and, unfortunately, those guards had to go collect numbers and x-es all over the place, so Church decided to move everything to the left for those too:

$5 (\backslash x. 2 \ x) = (2 \ 5 \ *) = 10$

Twi is, in all reality, just reversed lambda with lots of syntactic sugar on top. Also, in Twi guards have names so that they can write something down in their phone-books and call each other if they need help:

Code addTwo x = x 2+

So instead of writing the lambda symbol, you write their name, and instead of writing period, you write equals. So if one guard needs help from another, he can just find his name in his phone-book and call him:

Code addFour x = x addTwo addTwo

Also, a Twi guard can only fetch stuff from his own line, he cannot steal stuff from other lines:

Code addTwo x = x 2+

5 addTwo

Lines without guards, as well as other guards can however call guards and ask for help:

Code 5 addTwo

5 (x = x 2+)

(5 2 +)

5 2 +

7

Remember that the mathematical operators have invisible guards. Those guards are just like our guards, we just can't see them.

They are a bit like spies; our guards have their names in their phone-books, but nobody has ever seen them. But what if I do want to see them, what do they look like you ask? Well, we can try to find out!

We know that they take two things:

(\x y. \_\_\_\_)

But we have no idea what is happening inside; it's a bit of a mystery.

Let's try to pass it something and find out what is going on inside:

Code addTwo x = x 2+

addTwo x = x 2 (\x y. \_\_\_\_)

addTwo x = x (\2 y. \_\_\_\_)

addTwo x = (\2 x. \_\_\_\_)

Well, it did not really make us any wiser, but why did we have to give it that x? Our guard doesn't know what it is either, and it seems pretty pointless that our guard has to fetch it for the mysterious plus doesn't it?

Code addTwo = 2+

Since plus requires two things to give us an answer and we give it one, there's one thing left to fetch, simple as that, no need for us to do the hard work for the plus, we just leave the plus who've just eaten a two on the doorstep and wait for it to eat the next thing.

Code addTwo = 2+

5 addTwo

5 (2+)

5 2+

7

Similarly you can ditch the parameters from a lot of functions:

Code addTwo x = x 2+

addFour x = x addTwo addTwo

multThree x y z = x y z \* \* Code addTwo = 2+

addFour = addTwo addTwo

multThree = \* \*

That's pretty much the basics of the Twi language.

A syntax summary can be found here (there's a lot more sugar to it):

<https://dl.dropboxusercontent.com/u/69377299/TwiSite/index.html>

====Why it's awesome (ideas)====

Everything that is not outputted are removed at compile-time.

Twi is lazy. Basically, there are two different ways to compute lambda:

(2 2\*) (\x. x 2\*)

Method 1:

$(2^{2^*}) (\lambda x. x^{2^*})$   
4  $(\lambda x. x^{2^*})$   
 $(4^{2^*})$   
 $4^2$   
8

Method 2:

$(2^{2^*}) (\lambda x. x^{2^*})$   
 $((2^{2^*})^{2^*})$   
 $(2^{2^*})^{2^*}$   
 $2^{2^* 2^*}$   
 $4^{2^*}$   
8

My claim is that method 2 is better than 1. Why? Because you wait till the very last moment before you compute anything.

What if you didn't need to compute it in the first place? What if the answer is never used?

Twis does calculations on compile time. Say that you think it's more convenient to write  $5 \cdot 100^*$  instead of 500. Twis would happily accept both, and both would be compiled to 500. Same goes for everything else too. In theory, everything except input could be computed at compile-time; however, since you may be calculating  $\pi$  or something equally horrible, there is a limit to how many computations each operation can carry out for it to be done at compile-time.

Twis doesn't have types, everything is either a number or a list of numbers, which means that it boils down to math, which means that you could factorize it and simplify complex expressions.

Consider the following code:

```
5 2* put
5 3* put
```

Each line does some computations, and each outputs something. Both have to be computed, but since they are not really related to each other, you may calculate the answers in any order you like as long as you output them in the right order. And, because computers are fantastic at multitasking, Twis will actually calculate them simultaneously.

If a GPU is available in a long program, the GPU will calculate thousands of operations in the back of the programs simultaneously while the CPU raises through the code in the front towards the already calculated pieces of code.

The CPU is mostly used to connect the dots really, and the Twis counterpart of programs that you would have to wait a bit for could in many cases only be limited by how fast your CPU manages to output the answer.

Twis can lossily (you lose functions) be translated to binary lambda calculus to save space or for fast download. <http://homepages.cwi.nl/~tromp/cl/cl.html>

====Questions====

Is there an issue with having the GPU compute the program in the background, and if no, why have it not been done yet?

Other issues you can spot?

If you read the entire thing, did you find it difficult? Interesting? Did you get it at all?

Any suggestions? Feedback? Did you like the language?

**VirusTotal** - *July 28, 2014, 9:52 am*

TL; DR.

Btw good luck with your project dude

**Modesty** - *June 25, 2015, 12:06 pm*

(Jul 28, 2014 09:52 AM)VirusTotal Wrote: [\[CITE\]](#) TL; DR.

Btw good luck with your project dude

TL;DR: Check out how awesome lambda calculus is. I know this is an old thread, but people should be aware about the foundations of computation if they've got an interest in programming.

**Slachter** - *July 4, 2015, 1:46 pm*

Actually, this is really cool.

Keep it up!