Pyash: Humanity Fluent Software Language

Logan Streondj

February 13, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This is a human speakable programming language, geared for artificial general intelligence development.

1.1 Problem

1.1.1 Disglossia

Computer Languages

In order to program all levels of a modern computer, you need to know many different programming languages. Assembly/LLVM/SPIRV at the lowest level, C for system programming, C++/GTK/QT for graphical interface programming, Javascript/HTML/CSS/PHP/Perl for web programming, OpenCL/OpenMP/Pthread for parallel programming, bash/python/ruby/node for scripting, Java/C#/Lua for portable programming, ansible/chef/docker/kubernetes for administration, a host of different data storage formats XML/CSV/JSON/YAML/SQL and different documentation languages LaTeX/Doxygen/Markdown/TexInfo just to name a few. That doesn't even include statistics, audio, image and video processing languages.

Human Languages

You may have heard the story of Babel, and the story of Eve and the apple. The evidence goes back much farther, to Mitochondrial Eve, in thesecond last glaciation about 130 thousand years ago.

Mitochondrial Eve is the most successful mother, the mother all homo-sapiens share.

Eden was the great rift valley of Africa.

Mitochondrial Eve lived during times of famine where most women were likely too starved to be fertile. Eve was a clan leader and got enough food to reproduce, she had many daughters that became clan leaders.

Human language may have been perfected by Mitochondrial Eve, she helped her clan work together more efficiently than all the others during the population bottleneck, and thus came out the winner.

To this day we inherit our language brain centers from the mothers side.

Eve's daughters spread out, some went west, some, south, some east, some north.

Those that went South and East preserved clicks, like the Khoisan people of the kalahari. Those that went west into the jungle became the pygmies. Those that went north eventually became the farmers, the Bantu and peoples of the rest of the continents.

The language of Mitochondrial Eve can be reconstructed based on the common features of the oldest languages in the world, as well as our genetic predispositions to prefer certain forms of grammar.

The most common grammar form, and the one we are predisposed to is subject-object-verb (SOV), or head-final with postpositions and-or suffixes. Similar to Khoe (of the khoisan), Basque (the first homo-sapiens in Europe), Australian languages, Turkic (Central Asian), Uralic (North Eurasian),

Tibetan/Burmese (East Asian) and Proto-Indo-European (That conquered the world).

1.2 Paradigm

1.2.1 Easy to write bad code

In most, perhaps all contemporary languages it is easy for beginners to write bad code.

In assembly it is easy to write tangled spaghetti code. In C and C++ it is easy to have memory problems (buffer overflow, memory leaks, reading unassigned variables, etc) In Garbage collection languages it is easy to spend significant computer resources on allocating and deallocating memory. In Object Oriented languages it is easy to write unscalable code (any which uses objects). In functional programming languages it is easy to write memory bound code (non-tail recursive with lots of allocation and deallocation).

It often takes a lot of expertise to know the workarounds for the common programming traps, and even harder to apply them consistently.

1.2.2 Obsolete Non-Parallel Paradigms

Object Oriented, non-tail recursion and referentially opaque code are all obsolete considering that GPUs and parallel hardware are where processing power is growing the fastest.

It's easy to write bad code in many paradigms.

1.3 Inspiration

I was mentally projecting myself into a robot host body one day and realized that it would take a superhuman Artificial intelligence to be proficient in all of the languages and protocols of a modern computer and their interactions.

And I came to the realization that I wanted to be able to have access to all my knowledge and abilities with one language.

1.4 Answer

The answer I came up with is the speakable programming language.

1.4.1 Vocabulary

The root vocabulary was generated by taking the most frequently used thirty-eight thousand English words, translating them into the top thirty to forty human languages, and then removing words that were ambiguous and-or homophones.

This left a remainder of about eight thousand words, which were common to all languages and orthogonal (not overlapping in meaning).

This way you can use the root words of your preferred language to program, and they will be translated to all the other languages.

1.4.2 Grammar

Pyash currently uses Eve's grammar: SOV with postpositions and-or affixes is the grammar of Pyash the base language of Pyash.

Transferring it to other forms of grammar is fairly straight forward. And has been done with a former iteration of this language. That does however lead to a large number of variants.

So for the near future, will simply have Eve's grammar, and your preferred vocabulary for root words.

Many contemporary languages lost the nominative-accusative case distinction, and have grammar words which are used ambigiously. For example, the word ``with'' in English is used for comitative-case and instrumental-case.

So for the actual grammar words, have decided to go with abbreviated forms of the translations of glossing abbreviations. For example _com for comitative-case and _ins for instrumental case.

Because of knowledge bias, we'll have to work together to create documentation that is easy for beginners to understand.

1.4.3 Paradigm

The paradigm conforms to the JPL ten commandments[JPL10].

- Restrict all code to very simple control flow constructs - do not use goto statements, setjmp or longjmp constructs, and direct or indirect recursion.
- All loops must have a fixed upper-bound. It must be trivially possible for a checking tool to prove statically that a preset upper-bound on the number of iterations of a loop cannot be exceeded. If the loop-bound cannot be proven statically, the rule is considered violated.
- Do not use dynamic memory allocation after initialization.
- No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than about 60 lines of code per function.
- The assertion density of the code should average to a minimum of two assertions per function. Assertions are used to check for anomalous conditions that should never happen in real-life executions. Assertions must always be side-effect free and should be defined as Boolean tests. When an assertion fails, an explicit recovery action must be taken, e.g., by returning an error condition to the caller of the function that executes the failing assertion. Any assertion for which a static checking tool can prove that it can never fail or never hold violates this rule. (I.e., it is not possible to satisfy the rule by adding unhelpful `` assert (true) '' statements.)
- Variables must be declared at the smallest possible level of scope. All arrays must have a max-length variable, and bounds of all new index points must be checked before a read or write operation occurs. If an array is uninitialized, there must be an initialized-length variable also, so uninitialized data is not read accidentally.
- The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.
- The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions.
- The use of pointers should be restricted. Specifically, no more than one level of dereferencing is allowed. Pointer dereference operations may not be hidden in macro definitions or inside typedef declarations. Function pointers are not permitted.
- All code must be compiled, from the first day of development, with all compiler warnings enabled at the compiler's most pedantic setting. All code must compile with these setting without any warnings. All code must be checked daily with at least one, but preferably more than one, state-of-the-art static source code analyzer and should pass the analyses with zero warnings.

Additionally some OpenCL restrictions.

- no return values.
- input parameters are constants.
- output parameters are pointers.
- functions that don't interact with environment are referentially transparent.

Those should all be taken care of automatically, when compiling from Pyash to OpenCL C. So while other programming features may be available, it would take extra effort to enable the program to use them, and thus to write bad code.

Part I

Core Language

Chapter 2

Phonology

There are two scripts for the SPEL core-language, one based on URL-compatible ASCII and one based on IPA. If you are unsure of how to pronounce a letter, then simply copy paste the IPA letter into wikipedia which will give ample explanation.

Phonemes are based on the most popular distinctive ones on phoibles http://phoible.org/ parameters plus two clicks.

See table 2 for the ASCII, IPA and their description.

2.1 Notes

Alignment the `h' or /ʰ/ is a semi silent h /h/, and is used mostly for alignment purposes. All words when written in text are either 2 or 4 glyphs long. However some root and grammar words are three letters, thus they need alignment. For 3 letter roots of the form CVC (consonant vowel consonant) the h prefixes the word, turing it into hCVC, for 3 letter grammar words of the form CCV, the h is suffixes it, turning it into CCVh. A simple way to remeber this is that all words must comply with the CCVC or CV form. So if a three letter word is missing one of those C's then replace it with an `h' to get proper alignment.

Glottal stops glottal stop `.' is only used for foreign quotes, such as that of proper names, as they don't necessarily conform to alignment rules

Tones Tones `7' and `2', are mostly for low frequency words

Clicks Clicks `1' or `8' are used for temporary words and variables, especially useful to make short forms of compound words which are often used in a text or flock of people. Other options for short-forms are acronyms which must comply with the phonotactic rules of the language and be grammatically marked as acronyms, and initialisms, which are foreign quotes as they don't fit the phonotactic rules.

2.2 Contribution

Currently the phonology is pretty much finished, however if there are some compelling arguments then it may still be modified.

ASCII	IPA	Description	English
a	ä	central open vowel	arm
b	b	voiced bilabial plosive	ball
c	ʃ	unvoiced post-alveolar fricative	shout
d	d	voiced alveolar dental	door
e	ɐ̟	mid front unrounded vowel	enter
f	f	unvoiced labio dental fricative	fire
g	g	voiced velar plosive	great
h	ʰ	aspiration	happy
i	i	unrounded closed front vowel	ski
j	ʒ	voiced post-alveolar fricative	garage
k	k	unvoiced velar plosive	keep
l	l	lateral approximants	love
m	m	bilabial nasal	map
n	n	alveolar nasal	nap
o	o̟	mid back rounded vowel	robot
p	p	unvoiced bilabial plosive	pan
q	ŋ	velar nasal	English
r	r	alveolar trill	(Scottish) curd
s	s	unvoiced alveolar fricative	snake
t	t	unvoiced alveolar plosive	time
u	u	rounded closed back vowel	blue
v	v	voiced labio dental fricative	voice
w	w	labio velar approximant	water
x	x	velar fricative	(Scottish) loch
y	j ɻ	palatal approximant	you
z	z	voiced alveolar fricative	zoom
_	ʔ		glottal stop uh-oh
6	ə	mid central vowel	uh
7	˥	high tone	what?
2	˩	low tone	no!
1	|	dental click	tsktsk
8	‖	lateral click	winking click

Chapter 3

Grammar

3.1 Composition

those marked with asterisk are mandatory for phrase formation.

Phonology

Grammar Words are of the form CV or CCVH where C is consonant, H is /h/ and V is vowel.

Root Words are of the form HCVC or CCVC where C is consonant, H is /h/ and V is vowel.

sentence emotions evidentials mood*

noun phrase (root or pro-form)* suffix case*

verb phrase (root xor copula)* (aspect or tense)*

adjective phrase (root or pro-form)* suffix adjective-marker* case
case needs to be marked if adjective comes after the head noun, or could be confused as part of a different noun phrase.

adverbial phrase (root or pro-form)* suffix adverb-marker*

3.2 Grammar Tree

3.3 Noun Classes

abstract-gender for thoughts and ideas

animate-gender for more active things, animacy intensifier

anthropic-gender for anatomically human-like things

augmentative greater in size or intensity

collective-noun noun taken as a collection of things, turns mountain into moutain range, or trees into forest, or shrub into shrubland

common-gender male or female gender, hermaphrodites and ambigious gender, such as bigender, trigender, pangender,etc

diminutive less in size or intensity

dual-number two of the noun

inanimate-gender for less active things, animacy lowerer

feminine-gender female gender

masculine-gender male gender

- noun
  - pro-form
    * person-deixis
    * time-deixis
    * space-deixis
    * interior-deixis
    * surface-deixis
    * under-deixis
    * discourse-deixis
    * social-deixis
    * amount-deixis
    * state-deixis
    * interrogative
    * pro-phrase
    * pro-sentence
  - suffixes
    * proximity
      · proximal
      · medial
      · distal
    * number
      · singular
      · dual
      · trial
      · paucal
      · plural
      · multal
      · collective
      · distributive
      · inclusive
      · exclusive
    * classifier
      · name
      · number
      · length
      · mass
      · time
      · electric current
      · temperature
      · amount-of-substance
      · luminous-intensity
    * animacy
    * volition
    * gender
    * specificity
    * definiteness
    * quantifier
      · assertive
      · elective
      · universal
      · negatory
      · alternative
  - adjective marker
  - adverb marker
  - case
- verb
  - aspect
  - tense
- correlative conjunction
  - conjunction (and)
  - inclusive disjunction (and-or)
  - exclusive disjunction (xor)
- particle
  - sentence-final-particle
    * mood
  - sentence-semi-final particle
    * emotions
    * evidentials
- subordinator
  - genitive
  - relativizer
- interjection

Table 3.1: Grammar Tree

mineral-gender natural forces, rocks, bodies of water, etc

neuter-gender non-reproductive entities, including asexual

paucal-number small amount of something, 8 bit value $(0-255)$

number moderate amount of something, 16 bit value between 256 and 65 thousand.

plural-number large amount of something, 32bit value, between 65 thousand and 4 billion.

multal-number giant amount of something, 64bit value, between 4 billion and 18 quintillion.

rational-gender for entities that have self-selected goal-oriented communication, such as humans, spiritual entities, aliens and machine intelligence

singular-number one of the noun

trial-number a few or three of the noun

vegetal-gender for plants and instinct level animals, including power plants, solar panels, turbines, simple and complex machines, including primarily remote controlled ones.

zoic-gender for higher animals, humans, and robots that are capable of making decisions, and learning.

rational-gender for beings that together with zoic abilities are capable of establishing missions, and persuing them.

artistic-gender for beings that together with rational abilities, share the wisdom they have acquired.

research-gender for fields of study, such as mathematics, physics, biology, and other -ologies, generally pertaining to exploration and acquisition of knowledge.

vehicle-gender for vehicles, typically those that are primarily for the purpose of transportiation. Includes space-ships and small asteroids/comets.

locality-gender for places, including homes, areas, lakes, and countries. Includes large asteroids, and common locations of a corporation, for-example school.

planetary-gender For planemo, or planetary-mass objects which are rounded by their own gravity, and major locations of a corportion, for-example local school board.

star-gender For stars and central locations of a corporation, for-example provincial school board.

galaxy-gender For galaxies and major divisions of a corporation or virtual world, for-example department of education.

universe-gender For universes, corporations and virtual worlds, for example a government.

3.3.1 grammatical number

1. singular
2. dual-number
3. trial-number
4. paucal-number
5. plural-number

3.3.2 noun classes for relative adjustment

- diminutive
- augmentative
- inanimate-gender
- animate-gender

3.3.3 noun classes by animacy

this is based on a thirteen chakra system with animist world view.

0. abstract-gender
1. mineral-gender
2. vegetal-gender
3. zoic-gender
4. rational-gender
5. artistic-gender
6. research-gender
7. vehicle-gender
8. locality-gender
9. planetary-gender
10. star-gender
11. galaxy-gender
12. universe-gender

3.3.4 noun classes regarding reproductive attributes

- neuter-gender
- male-gender
- female-gender
- common-gender
- anthropic-gender

3.4 Tense

past-tense things that happened

hesternal-tense yesterday

recent-past-tense

remote-past-tense

present-tense now, things that are happening

hodiernal-tense today

future-tense things that will happen

crastinal-tense tommorrow

soon-future-tense

remote-future-tense

3.5 Aspects

atelic-aspect a
cumulative-reference process for
SPEL it is like signal processing,
at any point it is still processing,
perhaps for parallel processes

cessative-aspect for ending process
for SPEL exiting process
for hardware description language
falling edge

completive-aspect completely and thoroughly
finished
for SPEL finished with no errors

continuative-aspect process started but
not active
for SPEL idle processes

delimitative-aspect temporary process

frequentive-aspect repetitive process
for SPEL can be used for servers/daemons

gnomic-aspect general truths
for SPEL defining functions

habitual-aspect habitual process
for SPEL can be provided services or features

inchoative-aspect begining of process
for SPEL loading process
for Hardware Description Layer signal rising edge

imperfective-aspect for SPEL a process which is ongoing
any partial process

momentane-aspect for things that happen suddenly or momentarily, like power surges and lightning bolts. For instance a clock-tick could be momentane and frequentive.

perfective-aspect any whole process
for SPEL a process which has completed

progressive-aspect for active process
for SPEL for active processes

prospective-aspect for processes that happen after
for SPEL queued processes

retrospective-aspect for processes that happen before
for SPEL prerequisite processes

telic-aspect quantized process
for processes where any of the parts are not the whole, only taken together is it the whole.
for SPEL this is processes that require sequential components of a different kind.

3.6 Grammatical Mood

admonitive-mood warning, I warn you that
for SPEL error messages

affirmative-mood agreeingly, I agree that
for SPEL selection of correct output, as per reinforcement learning or genetic algorithms

apprehensive-mood fearfully, I fear that
for SPEL throwing exceptions

assumptive-mood assumingly, I assume that
for SPEL assert statements

conditional-mood if such and such
for SPEL conditional clauses

commissive-mood I commit to, I promise that
for SPEL setting calendar events, and personal virtue goals, also for unit-tests
for Parliment, to send motion to comittee

benedictive mood blessings, I wish the blessing that
for SPEL increasing priority of a process

deductive-mood deductively, I deduce that
to mark conclusions through deductive inference

deliberative-mood shall I?, do you think that?
for SPEL asking user input

deonitic-mood I should, I ought to, I plan that
for SPEL pseudo-code

delayed imperative in future do that
for SPEL, scheduled jobs

desiderative-mood I want to, I desire that
for SPEL near term goal setting

dubitative-mood doubtfully, sarcastically, I doubt that
for SPEL for inverse assert statements

inductive-mood inductively, I derive that
to mark conclusions through inductive inference

inferential-mood for things which are infered based on premises.

epistemic-mood perhaps, I consider it possible that

Table 3.2: Aspect Tree

- state
  - perfective-aspect
    - momentane-aspect
    - completive-aspect
  - imperfective-aspect
    - continuous-aspect
    - progressive-aspect
    - delimitative-aspect
- occurence
  - gnomic-aspect
  - habitual-aspect
- part of time
  - inchoative-aspect
  - cessative-aspect
- relative time
  - retrospective-aspect
  - prospective-aspect

<h3>Lexical</h3>

- composition
  - atelic
  - telic
  - stative-verb
- causal
  - autocausative-verb
  - anticausative-verb

eventive-mood in the event that
for SPEL event catchers

gnomic-mood generally, In general I believe that
for SPEL function declaration

hortative should, I urge that

imperative-mood you must, I command that
for SPEL, imperative programming sentence

imprecative mood curse, curse that
for SPEL decreasing priority of a process, also for setting up security measures, such as firewalls, honey pots and others

indicative-mood indicating the real, I indicate that
for SPEL variable declaration

interoggative-mood questioningly, I question that
for SPEL search queries

irrealis-mood unreal sentence, it isn't real that
for SPEL comments

jussive-mood tell them to, I command them that.
for SPEL issuing commands to remote location

necessitative-mood I need that
for SPEL listing of required libraries

optative-mood I wish that
for SPEL long term goal setting

potential-mood possibly, I consider it possible that
for SPEL try statements

permissive-mood I permit that
for SPEL setting and limiting priviliges, also for SPARK style contracts
for Parliament set limits of debate

precative-mood I request that
for SPEL making network requests and pull requests
for Parliment amendments

prohibitive-mood don't, I forbid that
for SPEL, blocking certain things, or ignoring certain inputs

propositive-mood I suggest that, I propose that
for Parliment as a main motion starter in deliberative discussion

realis-mood It is real that

sensory-evidential-mood evidence I've experienced tells me that
for Parliament or Court, to bring evidence before the assembly, also to mark premises in logical arguments.

subjunctive-mood unreal clause

speculative-mood speculatively, I guess that

volitive-mood desires, wishes or fears

hypothetical-mood For things which aren't necessarily true, but could easily be true, from the speakers perspective.
also for spel catch statements

3.7 participles

Participles are when you need to use a verb-form in a noun-phrase, typically as an adjective modifying a noun.

It follows the same form as a normal verb, except that instead of ending with a mood it ends with the adjective suffix 'ci' /Si/ or a case-ending if it acts as a noun.

The general form is tense-voice-aspect

Table 3.3: Grammatical Mood Tree

- realis
    - indicative
    - evidential
    - energetic
- irrealis-mood
    - deontic
        * commissive
            · permissive
            · prohibitive
        * directive
            · imperative
            · hortative
            · precative
            · necessitative
            · jussive
        * volitive
            · desiderative
            · optative
            · apprehensive
            · benedictive
            · imprecative
    - epistemic
        * interrogative
        * speculative
            · assumptive
            · dubitative
            · potential
        * inferential
            · hypothetical
            · inductive
            · deductive
    - would be
        * conditional
        * eventive

	active	passive
present	ra	rapyoh or pri7h
past	ki	kipyoh or tri7h
future	bi	bipyoh

Chapter 4

Dictionary

4.1 Prosody

4.2 Trochaic Rhythm

First syllable strongest, emphasis on odd syllables, grammar words always unemphasized.
http://wals.info/chapter/17

4.3 Espeak

Espeak unfortunately does not have trochaic rhythm support at this time (Feb 2017)

4.4 Semantics

If a sentence has a dative case, then it will return the whole sentence in the consequence reflector, otherwise it will return what would have gone into the dative case as the result.

4.4.1 Execution Model

knowledge is what is known in addition to builtins, input is a request or action, that can be fulfilled with what is known and builtins.

Ankh based execution model:

Knowledge is the base of the Ankh. The short-term memory is in the center or Aten. The wings are the interpreter and evolver. The head is any predictive algorithms.

The Aten stores the current program state. It is a small piece of ring memory that fits in the L1 cache. Most recent variable updates go there. Have to make extra effort to commit to long-term memory or knowledge which is outside the cache.

Wheras knowledge is for more long-term information, primarily functions, and immutable datasets.

4.4.2 Variable Addressing

To address a variable can say the name of the variable and ask for what is in it. for-example ``prifgina hwatka ri'' or ``variable -nom -acc what -acc -int'' or ``named variable is what?''. This will look backwards from current place in short term memory, checking if nominative case matches the named variable, then will give the result of the nominative clause.

If it is not found in the short-term memory, then it will look into through the knowledge, possibly through an index stored in the knowledge for large files.

Atomic Variable Updates

for atomic variable updates, the old variable location is only deleted after the new one is written. So if it quits in mid-state then there will be two declarations. To make sure the declaration is valid can write the declaration index last, and erase it first --- when not doing parallel read/write.

4.4.3 Declaration addressing

function statements can be loaded with unique variables into the code aten, so that the result of a function call will assign a unique variable, when that variable has been resolved then execution can continue.

that way can keep up with the weird asynchronous processors all processing statements in parallel.

each bit of local memory can have it's own local Aten, where it processes pure functions, and just commits to the global memory the variable results when it gets them.

Thus the worker that puts a declaration onto the code Aten must rename the variables to something unique, to minimize risk of collision with existing variables.

Can deal with multiple threads in the same cache space by having different topic phrases for them, or different process names.

the hardest part of that is probably coming up with unique names in an asychronous fashion... though maybe if they various workers are fed some part of a pseudo-random chain, can have reasonble assurances. can actually do concurrency without context switching, by simply having a different topic phrase for each concurrent process. so they will be processed independently of each other

variable code reuse generalization

in the interest of code-reuse and generalization, the program that does varible interpreting, for instance deontic mood, can resolve the values of the variables before getting the code-name. the destination variable can be put into the dative-case. for instance if there is no dative-case, then the accusative-variable would upgraded to dative-case.

basically this transformation process would generate a new sentence where all the inputs are resolved values, and only the output is a variable name. it then outputs the output variable declaration with it's new value.

SSA

2018-02-22 15:05:13 htafdwes also, my current architecture actually does add a declaration of a variable for every time the variable is updated, and adds it to the top of the cache, then removes the old declaration, in order to perform an atomic update. that's the only way I can see of doing asynchronous updates when you have multiple processors processing the same program. though yeah, I'm still wondering about the best way of garbage collecting old variables that won't be needed later, currently I'm thinking of just moving them farther out of the cache, but for very large programs and SSA that would be a lot of memory bloat. can do what you guys were saying about the shared pointers earlier. like can have a count of how many reference a variable in the benefactive-case, and when it drops to zero, then can remove it. in SSA mode, the benefactive-case would grow while the value is unknown, and shrink after the value is known.

2018-02-22 15:20:11 jrslepak hold on, why are garbage collection and SSA interacting at all?

2018-02-22 15:24:05 htafdwes uh, well I haven't completely figured out SSA, but basically when a worker adds a function to the code-queue, will have to create unique variables and interoperate with the calling program. so the caller will wait on the result based on one or more variables that will result from the call.

htafdwes the "garbage collection" would be something that a worker does, if they notice that they decrement the benefactive case of a variable to zero upon reading it.

Streams and Promises

To enable streams and promises, or variables that take a while to complete assembly, can use keywords like processing and finally to indicate whether a variable is currently being created or if it has completed in it's creation.

If the variable is being created in parallel though, then it would be in an inconssitent state like a bittorrent, the ending and some middle parts may finish before the begining.

There can also be a count of how many variables are expected, and can decrement based on that, so when it reaches zero the worker can update the state to finally.

Also for streams that have a certain buffer size, can have a circular buffer, with an indicator of the current location, maybe using locative case.

For flow-based-programming can have one or more functions listed in the vocative case, that will be activated to process new stream input. also good for "push" notifications.

4.5 Rank

The rank or priority can be designated, possibly as part of the topic case, using prefix to designate the priority.

normal priority things would have a certain amount of space between executable lines, that way if a high priority interrupt occurs, it can fill in the blanks, and thus have fewer spaces between its commands.

highest priority would have no spaces in between commands, and as it gets lower there would be more space between commands.

if have a global settings of the various rank process which are available can allocate each appropriately.

Also when a processor/worker stumbles upon a high-priority process, then it gets sensitized for the high-priority, and it will sniff around more to see if there might be high priority commands hidden amongst lower priority commands after it returns froma high priority command, if not, it will just continue with the next highest priority that it sniffs out.

4.5.1 Asynchronous reading and writing to buffer

all reads and writes happen via atomic-swap. so when a worker is reading a line then it swaps the index with 0xFFFF (positive-infinity) while it reads the line.

for the normal case of a htin fits within the 15 bytes, then while writing the worker can put positive-infinity as the index, updating that value last.

Though for those that have several to write. Can write a uint32t to the 0th place of the form 0xFFFE (negative-infinity) followed by number of arrays that are required, it would also be updated last. Then it would fill the following initial ones with positive-infinity unless it doesn't have enough room, in which case would have to unset the ones it did back to 0 and search for the next available space.

this way any workers that come across the allocation mark will skip ahead beyond it for reading or writing, and minimize risk of collisions.

An additional help would be to have a circular buffer increment. which can be used for those writing to the buffer, it would be a value that is read, and if it is before where the worker has written, then the worker will swap it with just after it has written. This may ease the case for workers looking for free space.

Once a variable is updated, or runs out of callers, then it is erased by the last worker that accesses it. similarly once a piece of code is successfully evaluated it is erased by the worker that successfully evaluated it. To erase it, all that has to happen is the index is converted to 0. Thus algorithms should look for the last point of the sentence based on the index, rather than based on the last physical location, which could be misleading with garbage data.

4.5.2 Relative-Pointers

yeah, I think I can do a relative-pointer -type name -case style, then either the initial worker that loaded the code would put in the relative pointer, or the first worker that attempts to evaluate it. main issue is that relative pointers aren't generally human readable. but if I keep the name then it would be. would get replaced by value upon assignment anyways.

4.6 Phrase Composition

verb create
  phrase create
  tense retrieve

Chapter 5

Music

So I've been interested in whistling and talking drum languages like Silbo and Yoruba.

As you may know Pyash has a tone system, however I've specifically made it so that the most common words don't need tone in order to be expressed.

I'm wondering if I should change it to be all tonal instead, so that can be used like a talking drum language of Yoruba, which has the same number of tones as Pyash (3).

They use 8 base-3 whistles to represent each word. 9 base-3 whistles would be enough to represent all possible Pyash words, if enumerated, however that would require a dictionary. though if use a 10 base 3 system then should be enough for spelling out words.

3^3 initial consonants (have 22 consonants, and 3 word types), 3^2 second consonants (have 8 second consonats), 3^2 vowels (have 6 vowels), 3^1 tones, 3^2 final consonants (have 8 finals), which is $(3+2+2+1+2=10)$.

Though may be able to do a contour system as used in Silbo, to represent consonants. dono though, that compositional trinary thing seems pretty easy.

can actually make it 4 contour tones, a 3 tone contour, 2 tone contour, 3 tone contour, 2 tone contour.

to help with parsing, the least common 4 consonants will be assigned the outgoing middle tone for the initial tone contour, since the majority of vowels end with a middle tone contour.

that way can make Pyash music, that humans and computers can make meaningful sense of. each word would be a diamond, with a focal note that is the medium tone ans they will dance around it within an octave range. the next word can have a different middle tone.

also one of the advantages of whistling languages is that humans can use them for long distance communication, such as 5 km, using whistles or drums.. and robots can use them underwater for long distance also.

x k g t d p b f v q 1 8 n m c j s z r w l y

Number	Tone	Initial	Second	Vowel	Tone	Final
0	000 ˩ ˩ ˩	m /m/	h /h/	u /u/	2 /˩/	x /x/
1	100 ˧˩ ˩ ˩	k /k/	x /x/	o /o/	MT /˩/	k /k/
2	200 ˧˩ ˩ ˧	y /j/	f v /f v/	a /ä/	7 /˧/	g /g/
3	010 ˨˩ ˩	t /t/	c j / /			p /p/
4	110 ˨˩ ˩	d /d/	s z /s z/	6 /ə		n /n/
5	210 ˧˩ ˩	p /p/	r /r/	3 /æ/		m /m/
6	020 ˨˧ ˨	b /b/	w /w/	e /e/		f /f/
7	120 ˧˩ ˨	f /f/	l /l/	4 /ɨ/		c /c/
8	220 ˧˩ ˧	v /v/	y /j/	i /i/		s /s/
9	001 ˩˨	q //				
10	101 ˧˨					
11	201 ˧˨					
12	011 ˨˩	1 //				
13	111 ˩					
14	211 ˧˩	8 //				
15	021 ˨˧					
16	121 ˧˨					
17	221 ˧˩	n /n/				
18	002 ˩˨	m /m/				
19	102 ˧˨	c //				
20	202 ˧˨	j //				
21	012 ˨˨	s /s/				
22	112 ˧˨	z /z/				
23	212 ˧˨	r /r/				
24	022 ˨˩	w /w/				
25	122 ˨˩	l /l/				
26	222 ˩	y /j/				

Part II

Instruments

5.1 Language Instruments

dlutge denoter returns available denotations for input words.

hlikryange into-translator converts analytic language to Pyash source code.

kfige encoder encodes pyash source code to bytecode.

lwonpromge evolutionary programmer evolves programs from pyash bytecode specification.

vlicge virtual machine interprets pyash bytecode.

ksiktfikge consequence reflecter a REPL, reflects the consequences of various statemnts.

hk6nge bytecode to C compiler compiles pyash bytecode to C source-code.

hlakryange outof-translator converts pyash bytecode to analytic language.

mlathlasxrisge command-line-userinterface a friendly command line environment for linux.

Dlutge helps find appropriate word for input in the Pyash dictionary, it is particularly useful if you are just learning a pyash variant for the first time.

Hlikryange is what you use when you have decided on all your words and are ready to convert to Pyash source-code. For analytic input with Pyash grammar it can be streamed. For other grammar types and conjugated forms it is file to file. Input is .language-code.txt output is .pyac.txt

Kfinge encodes Pyash source-code into bytecode which is how most other tools process it. Input is pyac.txt output is .pyac

vlicge interprets Pyash bytecode and returns the output or sideffects, can also output a program state dump into bytecode that can be translated out for easy debugging.

Ksiktfikge is an interactive line interpreter, encodes the source code, then runs it, and returns the result. Can operate standalone or with a server/client model.

hk6nge compilers Pyash bytecode into C source-code for portable compilation. Can then be compiled into Javascript or otherwise.

5.1.1 mlathlasxrisge

mlathlasxrisge or xris (pronounced Hris) for short is a friendly command line interface to POSIX systems.

Instead of having a bunch of differently named applications for various purposes, it elects to have sensible defaults for most things.

Opening files

Linux has xdg-open so can simply use that as default for opening files.

The defaults will have to be configurable via the shell, and the keyword for it would be plin or opening.

Translating files

can use convert (from image magick) to translate images, and avconv to translate audio files, maybe ffmpeg or mencoder for translating video files.

All of those defaults could be changed,

regular expressions

Can use pgrep for regular expressions, or can make up a pyash specific regular expression format, possibly based on https://simple-regex.com/

navigating directories

Directory hierarchy can be illustrated through the genetive-marker for example root's home's user's directory
   main issue would be translating the LSB to Pyac, so that all of File Hierarchy erarchy root and root directory of the entire file system hierarchy.

/ taproot zruk

/bin binary bvih

/boot awake hket

/dev body tcic

/etc configure kxik

/etc/opt kxikti

/etc/sgml kxikti

/etc/X11 kxikti

/etc/xml kxikti

/home house hcas

/lib library htek

/lib<qual>

/media multimedia ht6t

/mnt landing trun

/opt alternate kren

/proc mind hmas

/root administration dran

/run running slac

/sbin operating-system binaries ps6tbvih

/srv service swas

/sys operating-system ps6t

/tmp temporary tric

/usr applications pyen

/usr/bin application's binaries pyentibvih

/usr/include application's headers pyentihxe2k

/usr/lib application's library pyentihtek

/usr/lib<qual>

/usr/local application's locally pyentinyip

/usr/sbin pyentips6thvih

/usr/share sharing zran

/usr/src open-source hxus

/usr/X11R6 draw hmuk

/var variable prif

/var/cache speed hsot

/var/lib priftihtek

/var/lock locks priftikla7k

/var/log recorder kyi2t

/var/mail priftimailbox jlos

/var/opt priftikren

/var/run priftislac

/var/spool waiting priftityuc

/var/spool/mail priftityuctijlos

/var/tmp priftitric

5.2 ksiktfikge

The line interpreter can be used as a system shell.

When a line-interpreter starts, it tries to connect to a running server, if there is one. If not then it starts a server and connects to it -- unless it is in a sandbox/standalone mode. by default it is on the pyac port 59652.

A typical authentication goes:

- S: "hello"
- C: "hello"
- S: "me -nom hostname's user -acc name -rea you -nom what -acc name -inte"
- C: "me -nom hostname's user -acc name -rea"
- S: "you -nom secret -or salt code -ins verification -inte"

the server main thread listens for incoming connections. there is also a processing thread, which processes code and data.

when there is a new connection, then main splits it off into a thread. the connection thread adds data and code to the processing buffer, based on availability.

If processing buffer growing faster than being consumed then more threads are added, and-or GPU is enabled to process it.

Client can handle authentication for a user that is logged into the local machine, by saving the password or salted hash and using CHAP authentication.

client users/processes could have capabilities list.

5.3 Dlutna: Denotation Finder

inner form:
   ziprih_stranger_prihzina ziprih_guest_prihziku ziprih_unknown_prihziku ziprih_foreign_prihzika dlutli
   Chat asked:

user stranger

dlutna na "stranger" ka "guest", "unknown", xor "foreign" be denote ?

user foreign

dlutna foreign

user ask

dlutna na "ask" ka "hear", "please", xor "pray" denote ?

user asked

dlutna asked

dlutna na "ask" ka "hear", "please", "pray", xor "asked" may

user yeah

dlutna reform mweh

Part III

Compiler

Chapter 6

Specification

Pyash simple compile to OpenCL.

6.1 stages of compilation

1. natural language text (perhaps)
2. analytic language text
3. Pyash language text
4. Pyash encoded tiles
5. if is declarative specification then evolve imperative implementation
6. add imperative implementation function to library if not available
7. compile to (OpenCL) C with Pyash names
8. (OpenCL) C with analytic names (perhaps)

6.2 Method for implementation

In theory can use any language for implementation. Though ideally would be a version of C which is similar to the above, so it could then be recoded in Pyash.

6.3 answer verification

The agree debug library is OpenCL and holy ceremeny (pure function) compatible.

Ideally would have a way of listing many inputs and their corresponding outputs. If this could be fed to an OpenCL kernel that would be delicious.

The agree debug library can be the ``testing framework'' for Pyash programs. So each agree statement adds a line to the newspaper, after the program is complete it can list the statements in the newspaper, saying those are the tests that failed. Additionally could have a list of the number that have passed.

I'm thinking can save both the line number, and the amount that have passed in the first line of the newspaper. It can be an actual sentence, with two 16bit spaces for the values.

gzat na hnuc do lweh hnuc do mwah slak fa li

A newspaper until number with number succeeded.

Pyash	Pyash	C	file
kratta krathnimna li	cardinal top cardinal name nom rea	int main () {	cardinalname.c
swicta hnimna li	social top name nom rea	void name () {	cardinalname.c
hmasta hnimna li	mind top name nom rea	inline void name (); inline void name () {	librarycardinalname.h librarycardinalname.c
krathmasta hnimna li	cardinal mind top name nom rea	kernel void name () {	cardinalname.cl

htipdoyu txikka hciccu	ten num ins indexFinger acc down con	if (i < 0xA) {
zrundofi	0 num return	return 0;
fe	finally	}
hnimna tyindo cyah	name nom three num cop	name = 3;
txikna zrondo cyah	indexFinger nom zero num cop	i = 0;
htipdoyu txikka hciccu hyikdoyu plosliwa htekhromli	ten num ins indexFinger acc down con indexFinger acc one num ins plus rea and library program rea	for {;i < 0xA; ++i}{ libraryprogram ();}

A newspaper should be at least 16 sentences long, which is one page or 512 bytes, and less than or equal to 512 sentences, (32 pages), since that is the most that could fit in L1 memory with other processes.

6.4 Memory

There is no dynamic allocation of memory, only static, until further notice.

This is because historically dynamic allocation of memory has led to many memory leaks and other problems.

6.5 Control Flow

Inline control flow is supported

Here are some inline loops:

- sequence -ben produce -dat recipe -acc repeatedly -deo
- count -allative-case produce -dat recipe -acc repeatedly -deo

Inline conditionals:

- comparison -conditional truth -counterfactual-conditional -deo
- comparison -conditional truth recipe -counterfactual-conditional false recipe-dep -acc produce -nom -rea
- produce -nom -clause-tail comparison -conditional truth recipe -counterfactual-conditional false recipe-dep -acc produce -nom -rea
- comparison -con truth recipe -counterfactual-conditional neo comparison -con neo truth recipe -dep -acc produce -nom -rea

6.6 translate all independentClauses to C

Any independent-clause can be turned into C.

can be of the form:

```
sort1-case1-sort2-case2-verb-mood (sort1 name, sort2 name);
```

6.6.1 C Name Composition

For C, will need to include the types of the names in order to properly call functions, otherwise would have to have extra searching to locate which function is being refered to.

This will make it a bit like Navajo or Swahili, where the noun class will be mandatory. So we should have easy grammar words for them,

for names of things:

plu paucal-number 8bit

do number 16bit

pu plural-number 32bit

ml6h multal-number 64bit

ml6hhsosve multal-number sixteen vector, vector of 16 64bit values.

fe referrential, pointer

crih letter, char

crihfe letter referrential, char *

It seems I would only need a hash table lookup for operating on the GPU, seems like most of the other stuff can be done with a few conditionals.

Chapter 7

Operation Template

7.1 overview

7.1.1 translation

An English programmer writes English text.
An English encoder encodes the English text to the Pyash medium code.
A Chinese translator decodes the Pyash medium code into Chinese text.
A Chinese programmer writes Chinese text.
A Chinese encoder encodes the Chinese text into the Pyash medium code.

7.1.2 Compiler

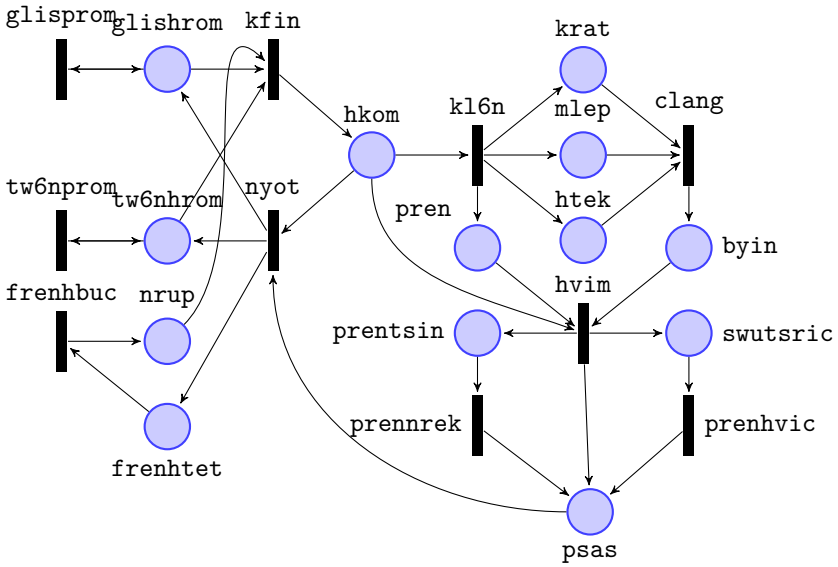
A code compiler from the medium code, to a cardinal ``.c'' file, library header file, and library ``.c'' files, as well as a kernel ``.cl'' file, and library file of intermediate code.

Clang compiler takes main and library ``.c'' files, and library header files, and produces an byin binary.

The byin binary operator sets up the constant stack, input data and makes writeable output data.

The host code starts the virtual machine kernel, and library kernels.

Figure 7.1: Compiler Petri Net



glisprom	English programmer
twynprom	Chinese programmer
glishrom	English program
tw6nhrom	Chinese program
frenhbuc	French user
frenhtet	French text
nrup	input
kfin	encoder
hkom	code
nyot	translator
kl6n	compiler
psas	produce or output
krat	cardinal file (main.c)
mlep	template file (lib.h)
htek	library file (lib.c)
pren	parallel file (kernels.cl)
hvim	runtime
prentsin	parallel knowledge (parallel da
prennrek	parallel workers
swutsric	holy recipes
prenhvic	parallel virtual-machine
psas	produce (output)

Chapter 8

Pyash Encoding

The virtual machine uses variable-length-instruction-word (VLIW), loosely inspired by head and tails instruction format (HTF). HTF uses VLIW's which are 128 or 256 bits long, however there can be multiple instructions per major instruction word.

8.1 VLIW's Head Index

The head is really a parse index, to show the phrase boundaries. In TroshLyash each bit represents a word, each of which is 16bits, when a phrase boundary is met then the bits flip from 1 to 0 or vice-versa at the phrase boundary word. index takes up the first 16bits of the VLIW. This would lead to 256bit (32Byte) VLIW's. The real advantage of the indexing occurs when there either multiple sentences per VLIW, or when there are complex sentences in the VLIW's. Having the VLIW's broken up into 32Byte chunks, makes it easier to address function entry points, which can be placed at the beginning of a VLIW. Can fit 16 VLIWS in a POSIX page, 128 VLIW's in a Linux page, so would only need 1 byte (8bits) for addressing functions that are within 1 page distance.

8.2 Word Compression

Now for the slightly more interesting issue of packing as many as 5 glyphs into a mere 16 bits. Why this is particularly interesting is that there is an alphabet of 32 glyphs, which would typically required 5 bits each, and thus 25bits in total. However the 16 bit compression is mostly possible due to the rather strict phonotactics of TroshLyash, as only certain classes of letters can occur in any exact place. The encoding supports 4 kinds of words, 2 grammar word classes and 2 root word classes. Where C is a consonant, T is a tone and V is a vowel, they are CVT, CCVT, and CVTC, CCVTC respectively.

8.2.1 CCVTC or CSVTF

I'll start with explaining the simplest case of the CCVTC word pattern. To make it easier to understand the word classes can call is the CSVTF pattern, where S stands for Second consonant, and F stands for Final Consonant. The first C represents 22 consonants, so there needs to be at least 5 bits to represent them. Here are the various classes

```
``C'' :``p'',``t'',``k'',``f'', ``s'',``c'',``x'', ``b'',``d'',``g'',``v'', ``z'',``j'',
    ``n'',``m'',``q'',``r'', ``l'',``y'',``w'',

``S'' ``f'',``s'',``c'',``y'', ``r'',``w'',``l'',``x'', ``z'',``j'',``v'',
```

``V'' ``i'',``a'',``e'',``o'',``u'',``6'',

``T'' ``7'',``_'',

``F'' ``p'',``t'',``k'',``f'', ``s'',``c'',``n'',``m''

, (can check the phonology page for pronunciation) C needs 5 bits, S would need 4 bits, however the phonotactics means that if the initial C is voiced, then the S must be voiced, thus ``c'' would turn into ``j'', ``s'' into ``z'' and ``f'' into ``v'', also none of the ambigiously voiced phonemes (l, m, n, q, y, w, r) can come before a fricative because they have a higher sonority, thus must be closer to the vowel. So S only needs 3 bits. V needs 3 bits T needs 2 bits and F needs 3 bits which is a total of 16 bits. 5+3+3+2+3 = 16 However there are other kinds of words also. we'll see how those work.

8.2.2 HCVTF

So here we have to realize that CVC or CVTC is actually HCVTF due to alignment. So what we do is make a three bit trigger from the first word, the trigger is 0, which can be three binary 0's, 0b000 3+5+3+2+3 = 16 H+C+V+T+C this does mean that now 0b1000, 0b10000 and 0b11000 is no longer useable consonant representation, however since there are only 22 consonants, and only 2 of those are purely for syntax so aren't necessary, so that's okay, simply can skip the assignment of 8, 16 and 24.

8.2.3 CSVT

This is similar to the above, except we use 0b111 as the trigger, meaning have to also skip assignment of 15, 23 and 31. $3+5+3+3+2=16$?+C+S+V+T

8.2.4 CVT

For this one can actually simply have a special number, such as 30, which indicates that the word represents a 2 letter word. $5+5+3+2+1$ $F+C+V+T+P$ what is PF P can be a parity-bit for the phrase, or simply unassigned.

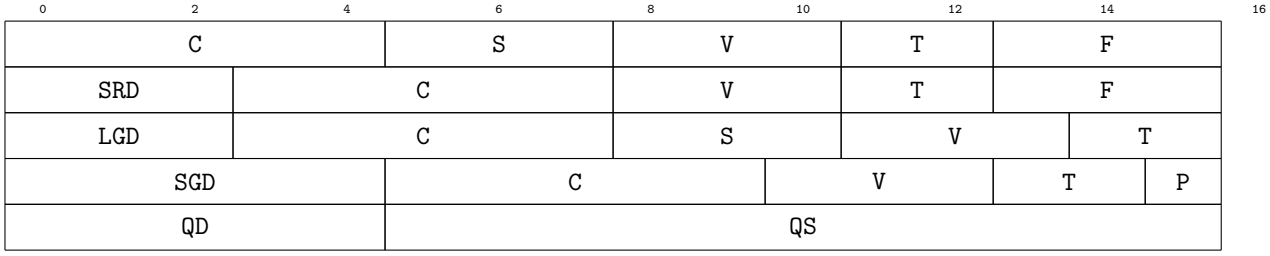
8.3 Quotes

Now with VM encodings, it is also necessary to make reference to binary numbers and things like that. The nice thing with this encoding is that we can represent several different things. Currently with the above words, we have 1 number undefined in the initial 5 bits. 29 can be an initial dot or the final one, can call the the quote-denote (QD), depending on if parser works forwards or backwards. Though for consistency it is best that it is kept as a suffix (final one), as most other things are suffixes. 5+3+8 = 16 Q+L+B QD has a 3 bit argument of Length. The Length is the number of 16bit fields which are quoted, if the length is 0, then the B is used as a raw byte of binary. Otherwise the B represents the encoding of the quoted bytes, mostly so that it is easier to display when debugging. The type information is external to the quotes themselves, being expressed via the available TroshLyash words. So in theory it would be possible to have a number that is encoded in UTF-8, or a string that is encoded as a floating-point-number. Though if the VM interpreter is smart then it will make sure the encoding is compatible with the type Lyash type, and throw an error otherwise.

8.4 Extension

This encoding already can represent over 17,000 words, which if they were all assigned would take 15bits, so it is a fairly efficient encoding. However the amount of words can be extended by increasing number of vowels, as well as tones. And it may even be possible to add an initial consonant if only one or two of the quote types is necessary. However this extension isn't likely to be necessary anytime in the near future, because adult vocabulary goes up to around 17,000 words, which includes a large number of synonyms. For instance the Lyash core words were generated by combining several different word-lists, which were all meant to be orthogonal, yet it turns out about half were internationally synonyms, so were cut down from around eight thousand to around four thousand words. It will be possible to flesh out the vocabulary with compound words and more technical words later on. Also it might make sense to supplant or remove some words like proper-names of countries.

8.5 Encoding Tidbit Overview



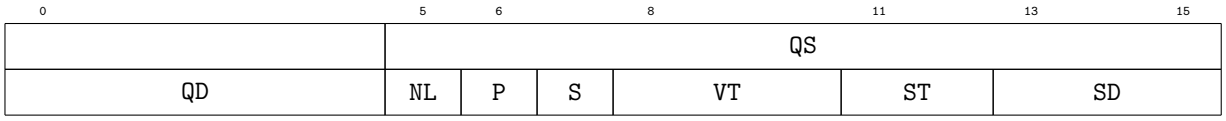
Legend C	Initial Consonant
S	Secondary Consonant
V	Vowel
T	Tone
F	Final Consonant
SRD	Short Root Denote
LGD	Long Grammar Denote
SGD	Short Grammar Denote
P	(optional) Phrase Parity Check tidbit
QD	Quote Denote
QS	Quote Sort 8.7

8.6 Table of Values

#	C	S	V	T	F
width	5	3	3	2	3
0	SRD	y /j/	i /i/	E	m /m/
1	m /m/	w /w/	a /ä/	MT /┤//	k /k/
2	k /k/	s z /s z/	u /u/	7 /┐/	p /p/
3	y /j/	l /l/	e /e/	_ /┘/	n /n/
4	p /p/	f v /f v/	o /o/		s /s/
5	w /w/	c j /ʃ ʒ/	6 /ə/		t /t/
6	n /n/	r /r/	4 /ɨ/ (U)		f /f/
7	LGD	x /x ɣ/	3 /æ/ (U)		c /ʃ/
8	SRO				
9	s /s/				
10	t /t/				
11	l /l/				
12	f /f/				
13	c /ʃ/				
14	r /r/				
15	LGO				
16	SRO				
17	b /b/				
18	g /g/				
19	d /d/				
20	z /z/				
21	j /ʒ/				
22	v /v/				
23	LGO				
24	SRO				
25	q /ŋ/				
26	x /ɣ/				
27	1 /|/				
28	8 /‖/				
29	QD				
30	SGD				
31	LGO				

	blank means out of bounds
E	Error signal
U	unused
MT	middle tone, no marking
QD	quote denote
SGD	short grammar word denote
SRD	short root word denote
LGD	long grammar word denote
SRO	short root word denote overflow
LGO	long grammar word denote overflow

8.7 Quote Sort



8.7.1 definitions

QS quote sort

QD quote denote

R referential

P pointed

S sequence

VT vector thick

ST scalar thick

SD sort denote

16 tidbit						
5 tidbit	1 tidbit	1 tidbit	1tidbit	3 tidbit	2 tidbit	3 tidbit
QD	referential	region	sequence	vector	scalar thick	sort denote
definitions						
0	literal	literal	literal	1	1 byte, _paucal_number	letter (s)
1	name	pointed	sequence	2	2 byte, _number	word (s)
2				4	4 byte, _plural_number	short-term (register)
3				8	8 byte, _multal_number	binary data
4				16		unsigned integer
5				U		signed integer
6				U		floating point
7				3		function

The quote denote is 5 bits long, leaving 11 bits. the next 2 bits is used to indicate bit thickness of quote scalar (s), the following 3 bits is used to indicate the magnitude of the vector (s), 1 bit for name or literal

letter l _letter

word word _word

phrase word _acc _phrase

sentence word _acc _rea _independent_clause

text

function

datastructure

named data type

unsigned integer one two three _number (291)

signed integer one two three _negatory_quantifier _num (-291)

floating_point_number two four _floating_point_num ten _bas one _neg _exponential _num (2.4)

fixed_point_number two _flo one _num (2.1)

rational one _rational three _num (1/3)

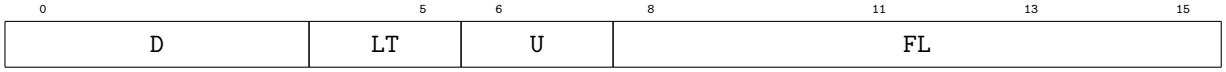
decimal number ten _bas one one _num (11)

hexadecimal number sixteen _bas eleven _num (11)

vector world _word _and _voc _word two sixteen word _vector (vector of 16 unsigned shorts each short containing a word, intialized to repeating sequence of ``hello _vocative_case'')

In the case of a referential, or variable name, the name can be (up to) four words long, that way it fits in a 64bit area --- similar to a 64bit address.

8.7.2 Sequence Extension



8.7.3 definitions

D number of sequence dimensions

LT sequence length thickness (same principle as scalar thick)

U unassigned

FL first dimension length, used only if scalar thickness is one byte.

If the sequence bit is flipped, then the byte following the quote short, has 4 bits for the number of dimensions, and 2 bits for the length of the scalar representing the length of each dimension.

For compression, if the length thickness is one byte then the second byte of the first short is used for the length.

Otherwise it is followed by the lengths of each array aligned by the length thickness.

For example if it was a three dimensional array where each dimension was under the length designated by a ushort,

then it would be quote-denote-short, sequence-short, first-demsnion-length-short, second-dimensions-l third-dimensions-length short, first dimension values, second dimension values, third dimension values.

One of the potential issues with storing them in the program, is that it would take a lot of space and would be more difficult to traverse large arrays.

So it may make sense to make the variable in a large variable pile area, and then simply have the pointer to it. So then it would be first-dimension-pointer, second-dimension-pointer, third-dimension-pointer. Which is more similar to the C implementation anyways.

8.7.4 Variable Pile

The variable pile is a statically allocated area in the code. All pointers in the code are relative to the local variable pile, effectively making it a sandbox. Attempts to read or write to pointer locations outside the local variable pile will return 0, or errors if possible.

Each recipe (function) has it's own variable pile for locally used variables.

The variable pile is primarily for storing variables that don't fit in a sentence. So any variable whose phrase wont fit in a standard short vector. Assuming 1 short for the quote, one for the index and one for the phrase, that leaves 13 for the value, or 12 if it is a sequence.

Also variables that are defined in a sentence should be inserted without breaking across a vector boundary, so if the phrase would break, then it should get it's own vector. This simplifies access to the phrase and variable contents. Breaks would only be acceptable for variables that are pointed to, so the pointers could span several vectors.

The variable pile is declared in terms of length of vectors of 16 shorts, and pointer locations point to the start of these vectors.

After a recipe slogan that contains a variable pile, is the variable pile slogan.

- one zero _num _allative_case pile _rea
- hyik zron do lweh mwak li

	0 source	1 way	2 destinatione	3 location
0 base	nominative-case	instrumental-case	dative-case	accusative-case
1 space-context (x)	ablative-case	prosecutive-case	allative-case	locative-case
2 genitive-case	partitive-case		possessed-case	associative-case
3 discourse-context	initiative-case	topic-case	terminative-case	vocative-case
4 social-context	causal-case	evidential-case	benefactive-case	comitative-case
5 surface-context (y)	delative-case	vialis-case	superlative-case	superessive-case
6 interior-context (z)	elative-case	perlative-case	illative-case	inessive-case
7 time-context (t)	initial-time	during-time	final-time	temporal-case

Table 8.1: grammtical-case number system

Then code can be read by starting afterwards.
The

8.7.5 Named Variable Example

8.8 Independent-Clause Code Name

Decided to make the independent-clause code name actually a universal hash, based on the sorts, cases, aspects and mood of the sentence. It's easier that way.
The grammatical cases can have a table to make it easy to identify them.

- literal literal literal 1234
- literal pointed literal address 1234
- literal literal sequence 5 ``hello''
- literal pointed sequence 5 address 1234
- name * * name

Part IV

Machine Intelligence

Chapter 9

Machine Programmer

recipe is a function in computer programming jargon.

holy recipe is a pure functionin computer programming jargon.

9.1 Oracle Based, or Active Synthesis

The interactive version of evolutionary programming is where a user starts a conversaion with the machine programmer.

First you would give a brief description about the purpose of this function, for documentation, and helping yourself narrow it down a little. Then you would make the function declaration with the input and output types, perhaps with some contractual constraints on the range of valid inputs and outputs.

At this point the machine programmer can check it's inventory of known programs to see if there are any matching programs, that have the same input/outputs, and similar names and-or description. If it finds a small amount of them, can ask if you'd like to use one of them as a basis.

After that user can give one or more input and output example and-or a logical description of what the output should be relative to the input. Can also offer some suggestions for which functions may be helpful in accomplishing the task.

At which point the machine programmer will start evolving functions that meet the specification and examples. If it is taking more than a few seconds user can start writing the program as they think it should be. the functions they thought would be necessary would be in comments, they can remove or add to them as necessary or uncomment them to include them in the code.

If the user decides that will need to make a new support function in order to achieve this task, then can save state of this development. and switch over to developing the supporting function.

If it evolves multiple programs that meet the full specification, but deviate from each other within the valid range of inputs. Then the machine programmer will ask which of them is prefered, or if neither then by giving the appropriate output for that particular input.

This process of refinement would occur until either the user is satisfied with the program or all the evolved programs having matching input and output pairs.

The machine programmer will then select one or more of the evolved programs to the library, based on what optimization have been set. For example, if size-optimization is enabled, then the smallest would be added, if speed-optimization is enabled then the fastest would be added, if fault-tolerance is enabled then two which call a maximally different set of functions would be added.

Got the idea from a book called "Program Synthesis" (2017), about a 4 hour read: https://www.nowpubli

9.2 Overview

A human programmer writes a recipe template, recipe suggestions and either provides a working recipe or sample input and output data.

An encoder encodes the recipe template and recipe suggestions into the intermediate representation (IR).

If the human programmer provides a working recipe, then the recipe profiler takes the recipe template IR and working recipe and generates the sample input and output data.

A population generator takes the recipe suggestions IR and input from /dev/random to create the population IR.

The population compiler converts the population IR into kernel or ``.cl'' files, one for each.

The population tester loads each population kernel, and streams the sample inputs through them, checking outputs for correctness, and produces the population fitness which includes fitness of all individuals.

The champion selector takes the fitness ratings, and the population IR, and outputs the champions.

The population mutator and recombiner takes the champions and recipe suggestions, then generates a new population IR.

An output generator takes the champions and outputs the best ones to a file.

9.3 input specification

The input specification consists of:

- the slogan of the recipe,
- description of the recipe,
- specification of the recipe,

9.3.1 Constraint Specification

The one implemented so far is one of the simplest, A form of constraint programing, commonly used in training contemporary AI.

- the training sequence
- a list of recipes to work with.

9.3.2 Specification example

```
specification -top recipe slogan -acc begin -rea
training sequence -top begin -rea
-quoted letter.A.letter -quoted -acc input -con
-quoted letter.a.letter -quoted -acc produce -rea
zero -num -acc input -con
zero -num -acc produce -rea
-fin
recipe sequence -top begin -rea
plus -rea
subtract -rea
-fin
```

```
regulation sequence -top begin -rea
-quoted letter.A.letter -quoted -abl down -con
input -nom produce -acc copula -rea
-fin
```

9.4 Evolutionary programming

multiple forms of evolution and optmial recipe discovery are used:

- random mutation (asexual reproduction)
- program inversion with backpropogation
- crossover (sexual reproduction)
- speciation
- neuro evolution
- specification generated tests
- adverserial coevolution
- neural networks

A computer program is similar to a recurrent neural net.

9.5 Ceremony produce

The produce of the evolutionary process contains the following items:

- recipe slogan
- best tests generated if any
- health achieved and recipe metrics
- unique identifier based on universal hash of recipe definition.
- required imports
- recipe definition including declared internal variables.

The recipe produce is a named recipe, appended to the file that has the same verb. If no such file exists then it is created.

This way various over-ride recipes will all be in the same file.

Automatic importing can happen by getting a list of all the verbs in the file and importing them. Though that could have a lot of problems, if there are name collisions or there are different providers of recipes.

9.5.1 produce example

```
holy -top letter tiny -nom tiny letter -num -dat letter -num -acc begin -dec
   metadata -top begin -rea
   -fin
   program recipe -deo
   -fin
```

Chapter 10

Codelet Bytecode Interpreter on GPU

10.1 Introduction

The plan is to make a next generation programming language for programming AGI. Constraints include using human grammar (linguistic universals), being compatible with genetic programming and maximize GPU usage, which is some of the cheapest and most underutilized processing power we have available.

In this paper only focusing on the maximizing GPU usage via the virtual machine which can run the intermediary representation. The intermediary code can also be compiled to C (host code), and OpenCL C (kernels), particularly for more traditional data-parallel applications.

But for the many processes which are not data-parallel, and instead have long computations such as compiling LaTeX files, those can be run through a virtual machine sitting implemented as an OpenCL kernel.

In fact the programming language like functional programming languages, encourages to keep all the input and output in the main function or monad, wheras the ones which are called are all referentially transparent.

This way can load-balance an application over as many cores as are available, including GPUs.

10.2 Previous Works

There are many works talking about getting OpenCL working inside a virtual machine, such as KVM[SPE:SPE2166][Gupta:2009:GGV:1519138.1519141][ratering2011accelerating], but that is completely different from having a VM running ontop of OpenCL.

One that sounds similar is ``OpenCL for Interpreter Implementation''[OpenCLInterpret] Though it compiles virtual machine bytecode to OpenCL kernels on CPU, so all it really shows is that compiled code runs faster on CPU than interpreted code on CPU.

Another similar one is ``Parallel Programming in Actor-Based Applications via OpenCL''[Harvey:2015:PP which talks about implementing actors while using OpenCL. Though actors are quite different from bytecode interpreters. And to use this would require translating code to an actor model, which may be difficult and not practical for many applications.

Instead I looked at highly parallel instruction set architectures, in particular the architecture used by the intermediary language is partially inspired by VLIW heads-or-tails architecture[Pan:2001:HTV

0	1	2	3	4				8				12			15
I	Q	C	P	Q	C	C	P	Q	C	C	C	C	P	V	M

Table 10.1: Codelet layout, composed of one ushort16, a 16bit phrase, a 32bit phrase, and 64bit phrase are demonstrated.

I Index
Q Quote denote
C Content or quoted value, number of ushorts it is composed of varies depending on bit length of value
P Phrase end word or grammatical-case
V Verb or command that operates on the phrases
M Mood word, or grammatical mood (end of sentence)
U Unassigned words after end of sentence

0	1			4				8				12			15
c	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p
1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1

Table 10.2: Index Overview

c Completion bit indicator, if equals 0 then ushort16 is only part of codelet
p Phrase or mood bit indicator, if is equal to completion bit, then a phrase word or mood word is here.

10.3 Operating Template

A codelet is a self-contained code module, the linguistic parallel is an independent clause familiarly known as a sentence.

Each codelet or independent-clause has several phrases for input and output, which are indexed by it.

The bytecodes or words that make up the phrases10.1 are all equal width.

In the Pyash implementation the codelets are each a ushort16 vector. If that is not enough to contain for instance a double16 constant, then the index (contained in the first ushort) indicates that it is not the final ushort16 (see table-10.1)

Each interpreting worker reads one of the ushort16s in the code array, if index starts with a partial then it skips to the next global id plus work group size short16. Though before it does, at the end of each evaluation all the workers synchronize to avoid race conditions.

If it is marked as final, it checks preceding indexes to get any extra short16's that make up the codelet, then evaluates it.

10.3.1 Memory Template

The program code is loaded into constant memory. The working memory is in an globally indexed local memory heap, and output is to global memory.

Each variable has a reference number in the referential phrase. The global index indicates if it has been set, and it's location in the local memory heap. The worker waits until all

0	1	2	3	4				8				12			15
I	Q	C	P	Q	C	P	V	M	Q	C	P	Q	C	P	V
I	M	Q	C	P	Q	C	C	P	Q	C	C	P	V	M	U

Table 10.3: Multi ushort16 Codelet layout, includes two conditional clauses, a 16bit phrase, a 32bit phrase, and 64bit phrase, are demonstrated.

I Index
Q Quote denote
C Content or quoted value, number of ushorts it is composed of varies depending on bit length of value
P Phrase end word or grammatical-case
V Verb or command that operates on the phrases
M Mood word, or grammatical mood (end of clause)
U Unassigned words after end of sentence

inputs are set before evaluating the codelet.

10.3.2 Control Flow

Control flow is managed through variables checked by codelet conditionals.

For example a comparison codelet sets an output variable, all the codelets whose execution requires the knowledge of that comparisons value check that it is set and that it passes their internal conditional before evaluating their codelet.

```
result = 1 > 2;
if (result == true) expression1();
if (result == false) expression2();
```

All workers check to see if the program is still running, to avoid hangs.

```
if (running == TRUE) result = 1 > 2;
if (running == TRUE && result == TRUE)
  expression1();
if (running == TRUE && result == FALSE)
  expression2();
```

See

If there are more layers of conditionals, then worker has to check them all.

```
if (running == TRUE && result == TRUE)
  result2 = 4 > 2;
if (running == TRUE && result == TRUE &&
    result2 == TRUE)
  expression3();
```

Can see an example of the layouf of a codelet that has multiple conditionals in figure 10.3

Variables

Each variable has to have a reference, along with it's value, and whether or not it has been set. For this there is a variable index, much like a hash-table. Each entry has:

- reference number
- fulfillment status (0 unset, 1 set)
- variable location

The lines that reference a value also should indicate if they are reading or writing the value. There should only be one line that writes the value, and the rest should read it.

Reading and writing can be implicit, because generally only the destination-class cases do any writing.

Line Evaluation

Each worker grabs a line

Program Queue

Instead of having a program counter, which says the current instruction, can instead have a program queue. This is valuable for function calls since they are outside the main program. If a worker happens upon a function call, it can add the contents of the function to the queue, so the next available workers will process it. Thus the program queue is more of a LIFO stack than a pipe.

However it doesn't have a stack pointer. The main program loads all the lines of a program in a global memory stack. Each worker also gets their own short stack.

To start work a worker atomic exchanges from the cardinal stack, if all the dependencies are met it does what it says. If a worker comes across a subroutine, they fill their short-stack with the lines of that subroutine. The top of a stack can contain an indicator saying that there are items in the stack.

If dependencies are unmet then can put the line back where it was found, and then check the tops of worker stacks --- starting with their own --- to see if any are non-empty, if there is one that is, then it proceeds to attempt the lines therin. If all the worker stacks are empty, it goes down the cardinal stack to the first non-empty operation.

If a line is successfully done, then it can either be put in a worker history stack for debugging and reversibility, and then it searches as usual, checking worker stacks and then cardinal one.

Branches

For parallel execution, any branch that remerges with the rest of the code would have to be a function call. So that variables wont be set unnecessarily.

Loops

All loops that don't break or return should simply be unrolled.

Otherwise there are two options for implementation,

The prefered option is simply to only have static length for loops, and always unroll them, having a variable to check if it has been broken.

Of course some for loop lengths are set at runtime for those can have a special looping variable which each codelet checks to see if it should continue. It can check after the

synchronization point to know if it should jump to the next codelet or continue evaluating this one.

Alternatively it could process everything, and

Jumps

Jumps may be necessary for some pieces of low-probability code. If there is a branch to a chunk of code, the branching worker can set a jump variable with the location (in constant memory) and length of the code. Then other workers would jump to evaluating their corresponding part of that code, or continue on the main code if they don't fit.

10.4 Speculation

It may not lead to significant performance gain, as it is interpreted rather than compiled, though it does inherently support superscalar execution, out-of-order execution, and speculative execution simply because all the codelets are executed in parallel, so may be quite fast, especially if implemented as a core architecure.

10.5 Conclusion and Further Work

This programming language may lead to increased usage of GPU's for a greater diversity of tasks. As of this writing I only have a basic prototype of the language, though with more time and effort it can become fully functional.