1st Annual Scryer Prolog Meetup

- Welcome to the meetup!
- Thanks to Christian & the HSD for hosting!
- Thanks also to the HSD & the Prolog Heritage Foundation for funding my visit

I will talk about:

- The year in Scryer Prolog
- Work I am now doing
- Future plans & ambitions

Documentation & Public Outreach

Scryer Prolog now has a proper homepage at `https://scryer.pl` (generously hosted by @aarroyoc)!

It hosts library documentation and a playground `https://play.scryer.pl`

Thanks also to @rujialiu for adding the 32-bit compilation + WebAssembly support to make this possible!

Version 0.9.3 is now out as github binary and on crates.io!

Community contributions in no particular order

Apologies to any one left out here! Community contributions of the last year are too numerous to list exhaustively.

- Atom table concurrency (@Skgland)
- Many improvements to clp(X) (X = $\mathbb{Z}$, X = $\mathbb{B}$) (@triska, @notoria)
- Github CI build system maintenance (@infogulch)
- HTTP library fixes and rewrites (@aarroyoc)
- `libffi` (@aarroyoc)
- The Djot markup language for documenting Prolog code (@aarroyoc) and much of the system documentation (@aarroyoc, @triska)

Community contributions in no particular order

- GMP $\rightarrow$ dashu arithmetic library migration (@fayeed, @lucksus)
- Adaptation of the Scryer Prolog engine for use as a client-facing library (@lucksus, @fayeed)
- dif/2 improvements (@bakaq)
- Docker image fixes (@gruhn)
- Test cases (@UWN, @notoria, @triska, @bakaq, @pmoura, @infradig, @flexoron, @haijinSk, @infogulch, @josd, @rujialiu)
- `#scryer` IRC channel on Libera chat (@rikardlang)

My contributions in no particular order

- Improved syntactic + ISO standard conformity
- `call/N` goal expansion caching
- Much better compilation of disjunctions (à la "Compiling Large Disjunctions")
- Lookahead indexing
- Introduction of stackless iterators (two-finger algorithm for heap marking and compaction, Deutch-Schorr-Waite for `acyclic_term/2`)
- Many bug fixes
- Golden Roll #1 on instruction templating and the dispatch loop (more to come)

In-progress work: Garbage Collection

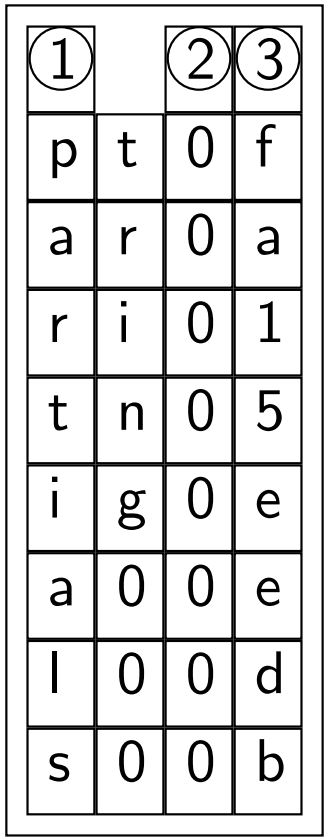
- Adapting the two-finger mark+compaction algorithm from the paper "Garbage Collection for Prolog Based on WAM"
- Partial strings embedded in the heap, requiring extension of GC to, e.g., truncate unreachable prefixes of strings
- Arena allocation (i.e. for constants of various sizes) will follow a BiBoP scheme, collected in tandem with the heap
- All this makes it much easier to track resources at the cost of implementation verbosity

In-progress work: Garbage Collection

- Some obstacles: `allocator_api` won't stabilize for years
- i.e. in many cases it would be nice to slot a custom allocator abstraction into common Rust data structures, e.g. `Vec<T, MyAllocator>`
- Escape hatch from RAII as the default memory management scheme in Rust
- Status quo forces us to write data structures from scratch or to extend existing libraries to allow custom allocations
- Some data types require finalization when freed, e.g. streams

In-progress work: Partial String Heap Section

- Heap layout is more like that in the WAM book: the heap, stack and trail are all allocated to a common block
- But, partial strings are allocated to a dedicated heap area
- A partial string: the character data & null characters until next 8-byte boundary (#1), scratch area for prefix minimization during GC (#2), heap address of tail (#3)



In-progress work: Short Atom Optimization

- Atoms consisting of 6-bytes or fewer in their UTF8-encoding will be stored in heap cells as immediate values never written to the atom table
- Many programming languages do this for strings already
- The atom table is thread-safe now, so the optimization also serves to reduce the frequency of table reads, mitigating synchronization overhead

In-progress work: Tag Bit Truncation and Simplification

The ‘HeapCellValue‘ tag space has been revised as follows:

Tag	Previous Tag Width	New Tag Width
Str	6	4
Lis	6	4
Var	6	4
AttrVar	6	4
StackVar	6	N/A
PStrLoc	6	N/A
PStrOffset	6	N/A
Cons	1	3
F64	6	3
Fixnum	6	3
Char	6	N/A
Atom	6	4
PStr	6	N/A
CStr	6	N/A
CutPoint	6	3

In-progress work: Further improved disjunction compilation

In the event of a branch like this:

```
(   var(ET0) ->
    error:instantiation_error(term_expansion/2)
;   ET0 = [_|_] ->
    term_expansion_list(ET0, ETH, ET0Tail),
    term_expansion_list(Terms, ET0Tail, ETT)
;   ETH = [ET0 | ET0Tail],
    term_expansion_list(Terms, ET0Tail, ETT)
)
```

it would make more sense to compute the (relatively simple) branch conditions in order of occurrence, jumping as necessary, from a single instruction controlling the dispatch, no choice point needed.

In-progress work: Code GC

- Since clauses can be dynamically asserted and retracted, resource reclamation is incomplete without code GC
- Scryer has supported incremental clause compilation for over 2 years now
- Compacting the `Instruction` data type also allows similar compaction tricks, i.e. with mark and forward bits
- Also must think of synchronization: the plan is to have multiple `MachineState` instances sharing a common code vector and atom table

Future work: JIT (or AOT?) Compilation

- The current plan is to compile WAM instructions using the Cranelift code generator (`https://github.com/bytecodealliance/wasmtime/tree/main/cranelift`)
- Many compilers innovations to be brought to bear on compiling WAM! Exciting buzzwords: SIMD, compute on the GPU, multithreaded compilation?
- GC must be implemented and proven first however; could benefit from a concurrent effort?
- Perhaps Prolog could itself be used to statically analyze Prolog programs and produce optimization hints?

Future work: Inlining common predicates

- Most common inefficiencies of the WAM are dereferencing terms and shuffling `HeapCellValues` around in registers
- Inlining is one strategy to mitigate both
- A handful of ISO standard predicates (semi-deterministic ones like `var/1`, `atomic/1`, etc) are already inlined
- `=/2` and many other system predicates are long overdue for inlining

Future work: Compilation speed-ups

- Bottleneck when compiling Prolog predicates to the WAM: translating heap terms to the `Term` intermediary type and back again
- Eliminating `Term` should produce a substantial speedup
- Also making the `Instruction` data type much smaller, or better yet, of variable size, much like what was done for `HeapCellValue`
- Compilation of Scryer Prolog as a Rust program is also a growing concern as use of Rust metaprogramming + code generation grows

Future work: Support for multi-threaded Prolog programs

- How to coordinate communications among threads?
- .. or synchronize reads/writes to the code vector efficiently?
- Several concurrency models to choose from

Future work: Writing more Golden Rolls

- Golden Rolls are documents describing Scryer Prolog's implementation details
- Meant to serve both as reference documents and incitement for people to contribute system code
- I am happy to hear subjects of interest from people!

Future work: Smaller scale improvements

- Arithmetic and constant instruction specialization
- Arithmetic performance generally could be greatly improved independent of native code compilation
- Multi-argument and deep JIT Indexing
- Clobbering the most common sequences of instructions into single instructions