Scalable Multi-Language Data Analysis on BEAM

The Erlang Experience

Jörgen Brandt

Humboldt-Universität zu Berlin

2016-11-24

Erlang Factory Lite Berlin 2016

About me

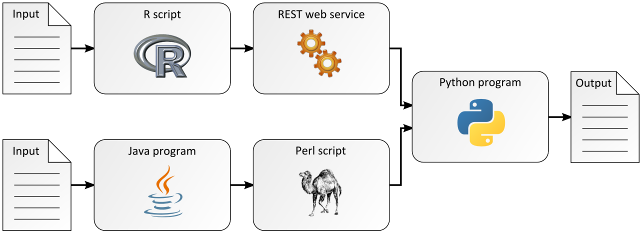
- PhD student at Humboldt (Berlin)
- Creator of Cuneiform workflow language
- Major area of application: Bioinformatics

Motivation

- Introduction to Cuneiform
- How to implement a large-scale data analysis PL
- How in Erlang?
- Why in Erlang?

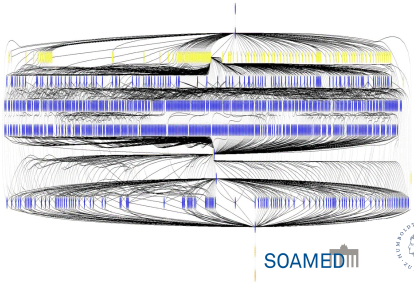
Scientific Workflow Systems

Scientific Workflow Systems



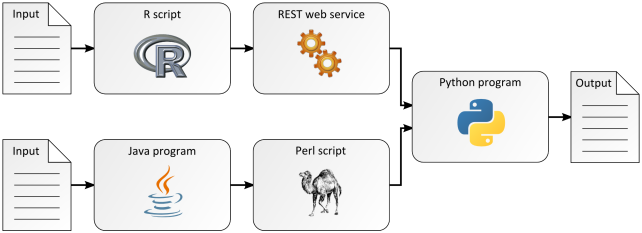
Workflows as DAGs

- Scientific Workflows are DAGs



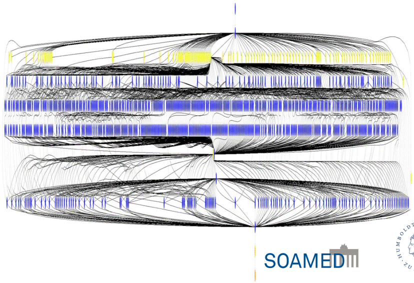
SOAMED

Scientific Workflow Systems

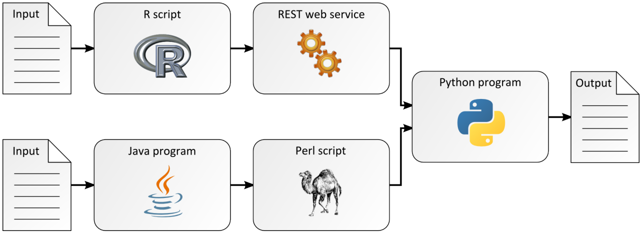


Workflows as DAGs

- Scientific Workflows are DAGs
- Nodes are tasks

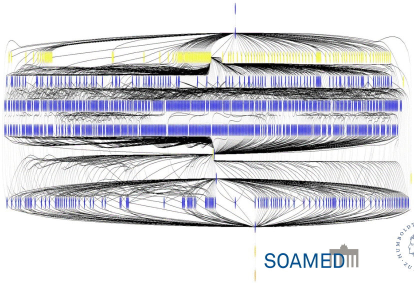


Scientific Workflow Systems

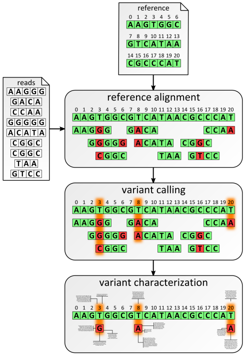


Workflows as DAGs

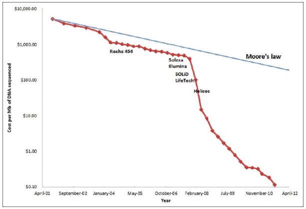
- Scientific Workflows are DAGs
- Nodes are tasks
- Edges are data dependencies

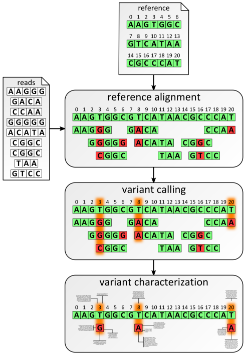


The Next Generation Sequencing use case

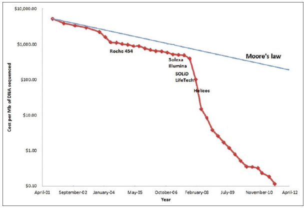


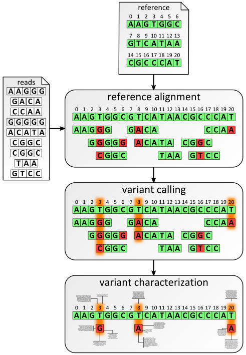
The Next Generation Sequencing use case



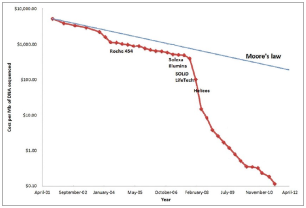


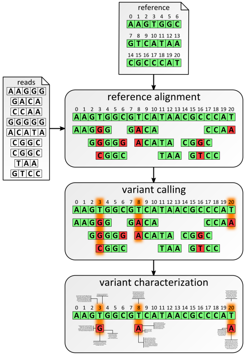
The Next Generation Sequencing use case

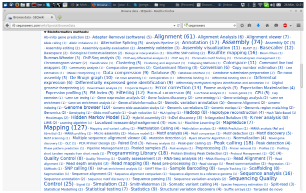




The Next Generation Sequencing use case





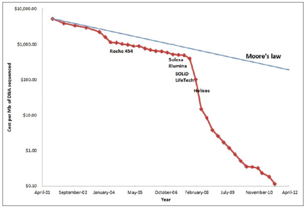


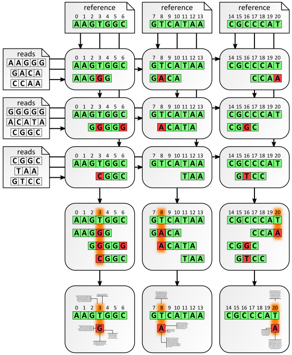
Browse data - SEQwiki - Mozilla Firefox
seqanswers.com/wiki/Special:BrowseData
seqanswers
26 Mär, 12:21

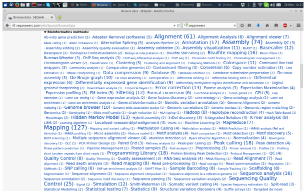
▾ Bioinformatics methods:
Ab-inito gene prediction (2) · Adapter Removal (software) (5) · Alignment (61) · Alignment Analysis (6) · Alignment viewer (7) · Allele calling (1) · Allelic imbalance (1) · Alternative Splicing (5) · Analysis Pipeline (2) · Annotation (17) · Assembly (74) · Assembly QC (3) · Assembly editing (2) · Assembly quality evaluation (2) · Assembly validation (3) · Assembly visualization (11) · BLAST (1) · Basecaller (12) · Basespace (2) · Biological Contextualization (2) · Biological interpretation (1) · Bisulfite SNP calling (2) · Bisulfite mapping (16) · Bloom filters (1) · Burrows-Wheeler (3) · ChIP-Seq analysis (3) · ChIP-seq differential analysis (1) · ChIP seq (1) · Chromatin motif finding (1) · Chromatogram management (1) · Chromatogram viewer (2) · Classification (1) · Clustering (5) · Clustering and alignment (1) · Collapsing Methods (1) · Colorspace (11) · Command line tool wrappers (3) · Community Analysis (1) · Comparative genomics (2) · Contaminant filtering (3) · Conversion (6) · Copy number estimation (3) · Cost estimation (1) · DNase I footprinting (1) · Data compression (9) · Database (5) · Database interface (1) · Database submission preparation (2) · De-novo assembly (3) · De Bruijn graph (10) · De novo Assembly (1) · Deduplication (1) · Differential Binding (1) · Differential binding sites (1) · Differential expression (6) · Differentially expressed gene identification (9) · Differentially methylated regions identification and annotation (1) · Digital genomic footprinting (2) · Downstream analysis (1) · Empirical Bayes (1) · Error correction (13) · Exome analysis (3) · Expectation Maximization (4) · Expression profiling (3) · FM-Index (5) · Filtering (12) · Format conversion (6) · Functional analysis (1) · Fusion genes (1) · GPU (5) · Gap extension (1) · Gene Set Testing (1) · Gene expression analysis (2) · Gene fusions discovery. (2) · Gene ontology (1) · Gene ontology analysis (2) · Gene set enrichment (1) · Gene set enrichment analysis (1) · General bioinformatics (2) · Genetic variation annotation (5) · Genome Alignment (2) · Genome Indexing (1) · Genome browser (10) · Genome wide association studys (1) · Genomic correlations (2) · Genomic overlaps (1) · Genomic region matching (2) · Genomics (2) · Genotyping (1) · Gibbs motif sample (1) · Graph reduction (1) · HLA typing (1) · Hadoop (8) · Haplotype reconstruction (4) · Hash Table Based (1) · Heatmaps (2) · Hidden Markov Model (13) · Hybrid assembly (2) · InDel discovery (3) · Integrated Solution (4) · K-mer analysis (8) · LIMS (2) · Learning algorithm (1) · Localized reassembly/realignment (4) · MCMC (1) · Machine Learning (2) · MapReduce (7) · Mapping (127) · Mapping and variant calling (1) · Methylation Calling (4) · Methylation analysis (1) · MiRNA Prediction (1) · MiRNA analysis (Ref and Ab-initio) (1) · MiRNA profiling (1) · Micro assembly (2) · Mixture model (1) · Motif analysis (4) · Motif comparison (1) · Motif detection (2) · Motif discovery (5) · Motif scanning (1) · Multiple sequence alignment viewer (4) · Mutation detection (2) · Myers Bitvector Algorithm (1) · Normalization (4) · Novel gene discovery (1) · OLC (1) · PCR Primer Design (2) · Paired End (3) · Pathway analysis (1) · Peak-pair calling (2) · Peak calling (18) · Peak detection (4) · Phase pattern prediction (1) · Pipeline Management (2) · Pooled samples (5) · Post-analysis (1) · Preprocessing (3) · Primer removal (1) · Profiles (1) · Profiling short tandem repeats from short reads (1) · Programming Library (18) · Protein Binding Peak Detection (2) · Protocol Management (1) · QC (4) · Quality Control (8) · Quality Trimming (1) · Quality assessement (3) · RNA-Seq analysis (4) · RRNA filtering (1) · Read Alignment (7) · Read alignment (1) · Read depth analysis (3) · Read mapping (8) · Read pre-processing (5) · Read storage (1) · Read summarization (2) · Regression. (1) · SAMtools (1) · SNP calling (6) · SNP set enrichment analysis (1) · SWIFT Filter (1) · Sample Barcoding (8) · Sample Tracking (1) · Scaffolding (8) · Segmentation (1) · Sequence alignment (2) · Sequence alignment comparison (1) · Sequence alignment to a reference genome (1) · Sequence analysis (16) · Sequence annotation (2) · Sequence motif discovery (1) · Sequence parsing (3) · Sequence variation analysis (2) · Sequencing Quality Control (25) · Signal (1) · Simulation (12) · Smith-Waterman (3) · Somatic variant calling (4) · Species frequency estimation (1) · Split-read (3) · Statistical Modelling (2) · Statistical testing (7) · Statistics (9) · Structural variation discovery (4) · Suffix arrays (2) · Targeted de novo

SOAMED

The Next Generation Sequencing use case







Browse data - SEQwiki - Mozilla Firefox
seqanswers.com/wiki/Special:BrowseData

▾ **Bioinformatics methods:**
Ab-inito gene prediction (2) · Adapter Removal (software) (5) · Alignment (61) · Alignment Analysis (6) · Alignment viewer (7) · Allele calling (1) · Allelic imbalance (1) · Alternative Splicing (5) · Analysis Pipeline (2) · Annotation (17) · Assembly (74) · Assembly QC (3) · Assembly editing (2) · Assembly quality evaluation (2) · Assembly validation (3) · Assembly visualization (11) · BLAST (1) · Basecaller (12) · Basespace (2) · Biological Contextualization (2) · Biological interpretation (1) · Bisulfite SNP calling (2) · Bisulfite mapping (16) · Bloom filters (1) · Burrows-Wheeler (3) · ChIP-Seq analysis (3) · ChIP-seq differential analysis (1) · ChIP seq (1) · Chromatin motif finding (1) · Chromatogram management (1) · Chromatogram viewer (2) · Classification (1) · Clustering (5) · Clustering and alignment (1) · Collapsing Methods (1) · Colorspace (11) · Command line tool wrappers (3) · Community Analysis (1) · Comparative genomics (2) · Contaminant filtering (3) · Conversion (6) · Copy number estimation (3) · Cost estimation (1) · DNase I footprinting (1) · Data compression (9) · Database (5) · Database interface (1) · Database submission preparation (2) · De-novo assembly (3) · De Bruijn graph (10) · De novo Assembly (1) · Deduplication (1) · Differential Binding (1) · Differential binding sites (1) · Differential expression (6) · Differentially expressed gene identification (9) · Differentially methylated regions identification and annotation (1) · Digital genomic footprinting (2) · Downstream analysis (1) · Empirical Bayes (1) · Error correction (13) · Exome analysis (3) · Expectation Maximization (4) · Expression profiling (3) · FM-Index (5) · Filtering (12) · Format conversion (6) · Functional analysis (1) · Fusion genes (1) · GPU (5) · Gap extension (1) · Gene Set Testing (1) · Gene expression analysis (2) · Gene fusions discovery. (2) · Gene ontology (1) · Gene ontology analysis (2) · Gene set enrichment (1) · Gene set enrichment analysis (1) · General bioinformatics (2) · Genetic variation annotation (5) · Genome Alignment (2) · Genome Indexing (1) · Genome browser (10) · Genome wide association studys (1) · Genomic correlations (2) · Genomic overlaps (1) · Genomic region matching (2) · Genomics (2) · Genotyping (1) · Gibbs motif sample (1) · Graph reduction (1) · HLA typing (1) · Hadoop (8) · Haplotype reconstruction (4) · Hash Table Based (1) · Heatmaps (2) · Hidden Markov Model (13) · Hybrid assembly (2) · InDel discovery (3) · Integrated Solution (4) · K-mer analysis (8) · LIMS (2) · Learning algorithm (1) · Localized reassembly/realignment (4) · MCMC (1) · Machine Learning (2) · MapReduce (7) · Mapping (127) · Mapping and variant calling (1) · Methylation Calling (4) · Methylation analysis (1) · MiRNA Prediction (1) · MiRNA analysis (Ref and Ab-initio) (1) · MiRNA profiling (1) · Micro assembly (2) · Mixture model (1) · Motif analysis (4) · Motif comparison (1) · Motif detection (2) · Motif discovery (5) · Motif scanning (1) · Multiple sequence alignment viewer (4) · Mutation detection (2) · Myers Bitvector Algorithm (1) · Normalization (4) · Novel gene discovery (1) · OLC (1) · PCR Primer Design (2) · Paired End (3) · Pathway analysis (1) · Peak-pair calling (2) · Peak calling (18) · Peak detection (4) · Phase pattern prediction (1) · Pipeline Management (2) · Pooled samples (5) · Post-analysis (1) · Preprocessing (3) · Primer removal (1) · Profiles (1) · Profiling short tandem repeats from short reads (1) · Programming Library (18) · Protein Binding Peak Detection (2) · Protocol Management (1) · QC (4) · Quality Control (8) · Quality Trimming (1) · Quality assessement (3) · RNA-Seq analysis (4) · RRNA filtering (1) · Read Alignment (7) · Read alignment (1) · Read depth analysis (3) · Read mapping (8) · Read pre-processing (5) · Read storage (1) · Read summarization (2) · Regression. (1) · SAMtools (1) · SNP calling (6) · SNP set enrichment analysis (1) · SWIFT Filter (1) · Sample Barcoding (8) · Sample Tracking (1) · Scaffolding (8) · Segmentation (1) · Sequence alignment (2) · Sequence alignment comparison (1) · Sequence alignment to a reference genome (1) · Sequence analysis (16) · Sequence annotation (2) · Sequence motif discovery (1) · Sequence parsing (3) · Sequence variation analysis (2) · Sequencing Quality Control (25) · Signal (1) · Simulation (12) · Smith-Waterman (3) · Somatic variant calling (4) · Species frequency estimation (1) · Split-read (3) · Statistical Modelling (2) · Statistical testing (7) · Statistics (9) · Structural variation discovery (4) · Suffix arrays (2) · Targeted de novo

Cuneiform is ...

Functional Language for Large-Scale Data Analysis
implemented in Erlang

- Different from systems like Spark
  - Standalone syntax, no fluent API in Scala
  - Integration of foreign PLs
- Different from distributed workflow languages like Snakemake
  - Reduction of functional expression, no static dependency graph

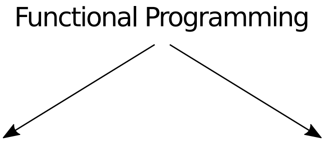
The main idea

Functional Programming

$+$

Foreign Language Interfacing

Functional Programming



- Very expressive
- Natural operations on lists (map, . . .)
- Natural iteration (recursion)

Gain

- General data analysis

Functional Programming

Functional Programming

- Very expressive
- Natural operations on lists (map, . . .)
- Natural iteration (recursion)

Gain

- General data analysis

- Parallelize independent sub-expressions
- Lazy Evaluation

Gain

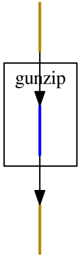
- Automatic Parallelism

Cuneiform Example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{
  out=output.file
  gzip -c -d $gz > $out
}*
```

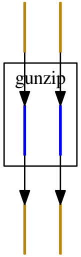
Cuneiform Example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{
  out=output.file
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz'
);
```



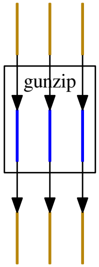
Cuneiform Example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{
  out=output.file
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz' 'myarchive2.gz'
);
```



Cuneiform Example

```
deftask gunzip( out( File ) : gz( File ) )in bash *{
  out=output.file
  gzip -c -d $gz > $out
}*

gunzip(
  gz: 'myarchive1.gz' 'myarchive2.gz' 'myarchive3.gz'
);
```

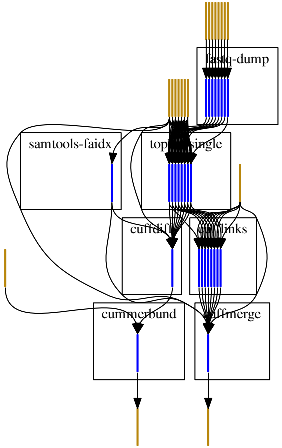


Workflow Implementations Available

`cuneiform-lang.org/examples`

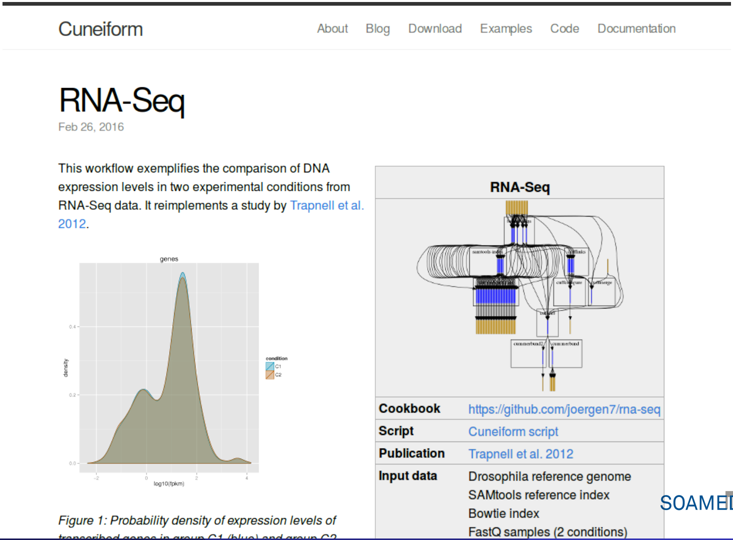
Available example workflows:

- Variant Calling (Varscan)
- Methylation
- RNA-Seq
- Variant Calling (GATK)
- etc ...



Example: RNA-Seq

`cuneiform-lang.org/examples`



Cuneiform   About   Blog   Download   Examples   Code   Documentation

RNA-Seq

Feb 26, 2016

This workflow exemplifies the comparison of DNA expression levels in two experimental conditions from RNA-Seq data. It reimplements a study by Trapnell et al. 2012.

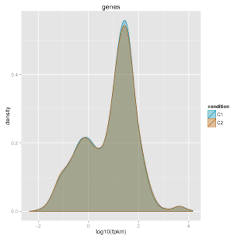
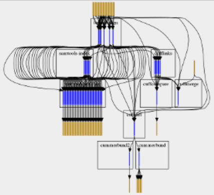


Figure 1: Probability density of expression levels of transcribed genes in group C1 (blue) and group C2

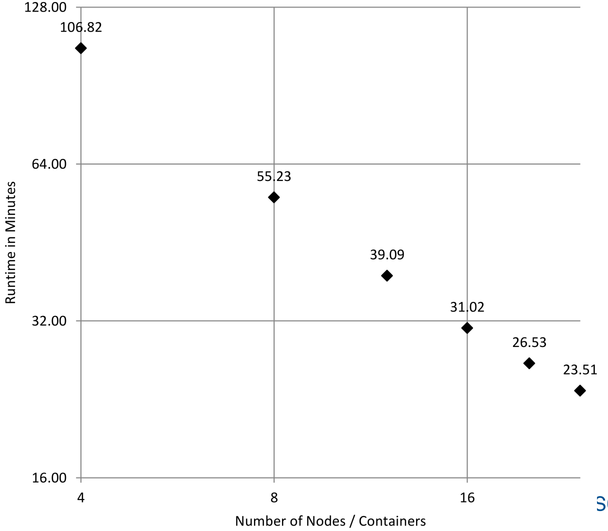
RNA-Seq



Cookbook	https://github.com/joergen7/rna-seq
Script	Cuneiform script
Publication	Trapnell et al. 2012
Input data	Drosophila reference genome SAMtools reference index Bowtie index FastQ samples (2 conditions)

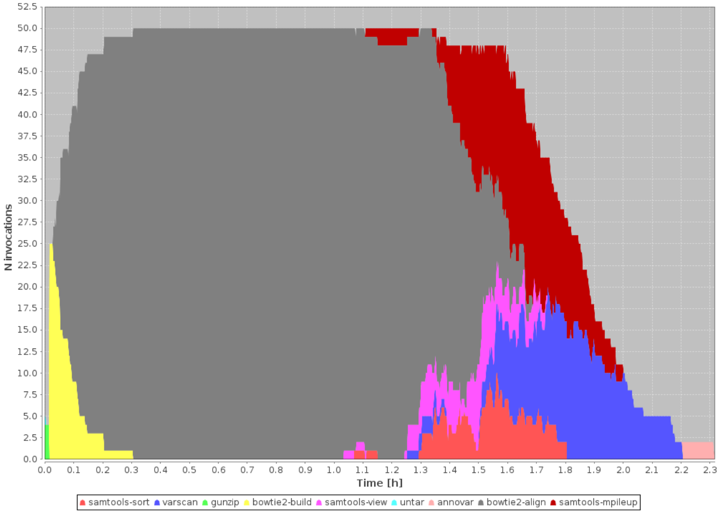
Scaling with cluster sizes

Scale-out with Hadoop for Variant Calling on 24 machines

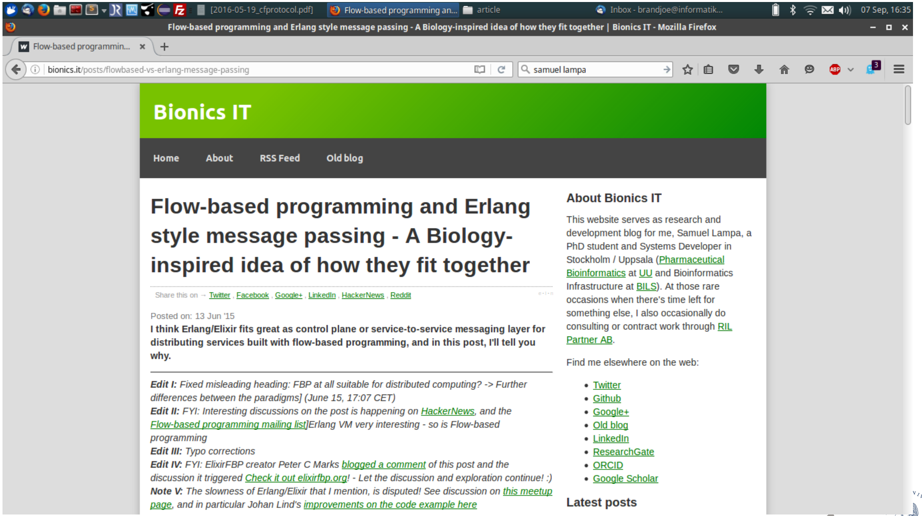


Time spent in workflow tasks

Variant Calling on 50 CPUs



Biology-Inspired Analogy



Flow-based programming and Erlang style message passing - A Biology-inspired idea of how they fit together | Bionics IT - Mozilla Firefox

bionics.it/posts/flowbased-vs-erlang-message-passing

Bionics IT

Home About RSS Feed Old blog

Flow-based programming and Erlang style message passing - A Biology-inspired idea of how they fit together

Share this on → Twitter , Facebook , Google+ , LinkedIn , HackerNews , Reddit

Posted on: 13 Jun '15

I think Erlang/Elixir fits great as control plane or service-to-service messaging layer for distributing services built with flow-based programming, and in this post, I'll tell you why.

Edit I: *Fixed misleading heading: FBP at all suitable for distributed computing? -> Further differences between the paradigms] (June 15, 17:07 CET)*
Edit II: *FYI: Interesting discussions on the post is happening on HackerNews, and the Flow-based programming mailing list]Erlang VM very interesting - so is Flow-based programming*
Edit III: *Typo corrections*
Edit IV: *FYI: ElixirFBP creator Peter C Marks blogged a comment of this post and the discussion it triggered Check it out elixirfbp.org! - Let the discussion and exploration continue! :)*
Note V: *The slowness of Erlang/Elixir that I mention, is disputed! See discussion on this meetup page, and in particular Johan Lind's improvements on the code example here*

About Bionics IT

This website serves as research and development blog for me, Samuel Lampa, a PhD student and Systems Developer in Stockholm / Uppsala (Pharmaceutical Bioinformatics at UU and Bioinformatics Infrastructure at BILS). At those rare occasions when there's time left for something else, I also occasionally do consulting or contract work through RIL Partner AB.

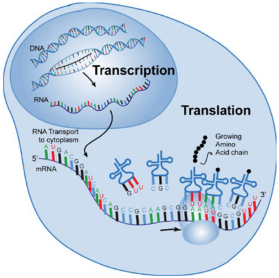
Find me elsewhere on the web:

- Twitter
- Github
- Google+
- Old blog
- LinkedIn
- ResearchGate
- ORCID
- Google Scholar

Latest posts

Biology-Inspired Analogy

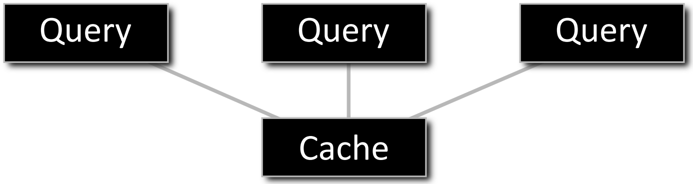
- Large-scale data analysis systems as 2-tier system:
  - Algorithm-level (fast, local)
  - Workflow-level (organizational, distributed)
- Analogy to the cell:
  - DNA transcription (fast, local)
  - Cell-to-cell signaling (organizational, distributed)



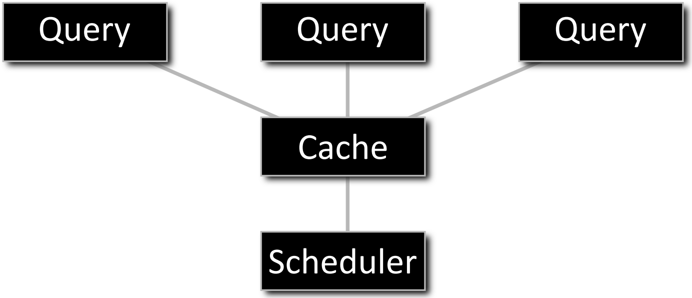
Erlang good fit for organizational part

Distributed Workflow System Architecture

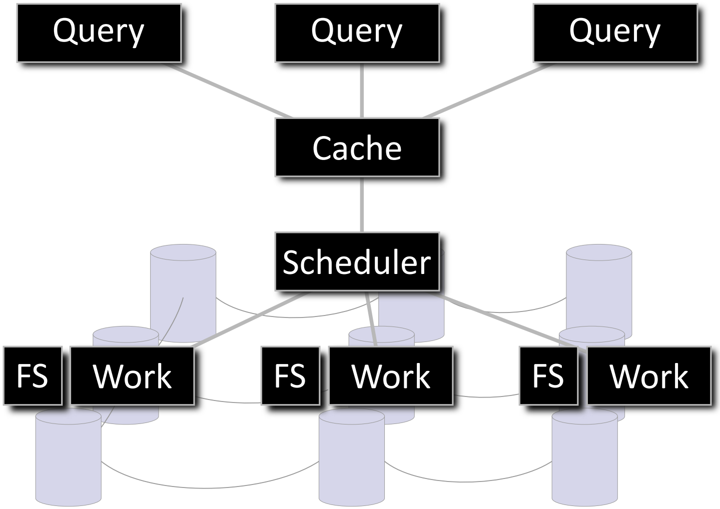
Distributed Workflow System Architecture



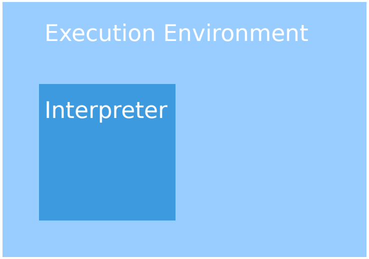
Distributed Workflow System Architecture



Distributed Workflow System Architecture



Two Modeling Challenges (i)



Interpreter

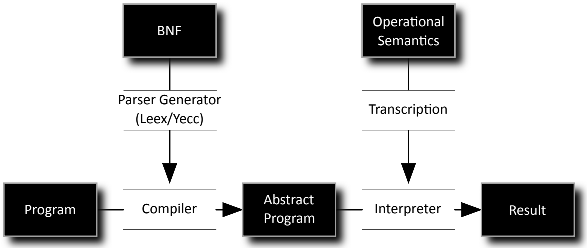
- Reduction of query expression

Execution Environment

- Distributed System

How to model programming languages?

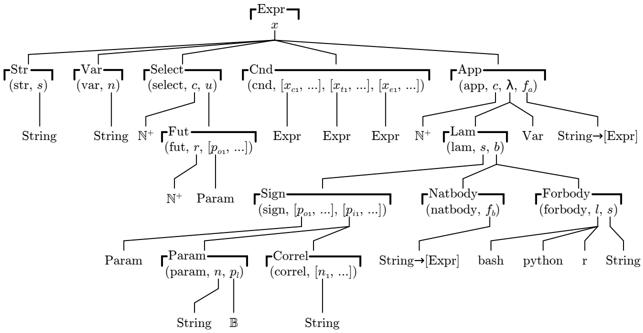
Modeling the Cuneiform Interpreter



- Parser is generated from BNF
- Interpreter is transcribed from Operational Semantics

Abstract Syntax

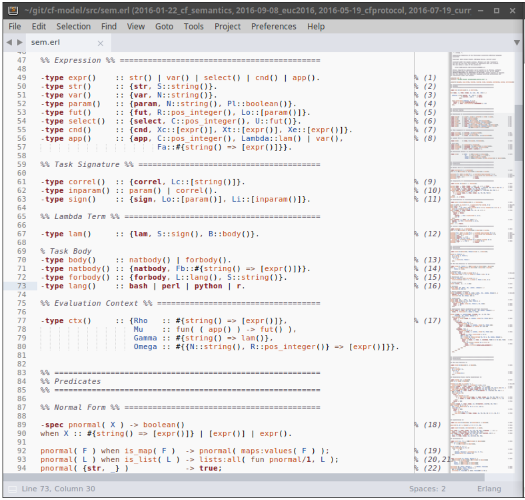
An expression in Cuneiform is . . .



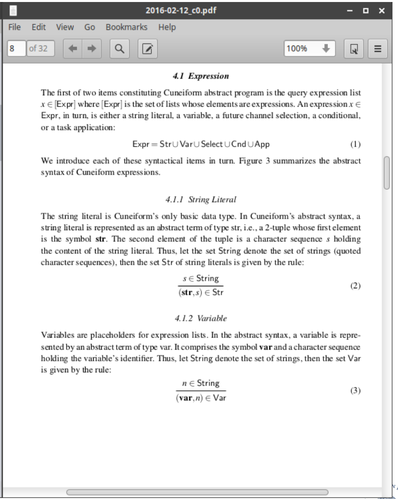
- Expressions can contain other expressions
- Semantics define how expressions are reduced

$$x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x^*$$

Implementing an Operational Semantics in Erlang


~/git/cf-model/src/sem.erl (2016-01-22_cf_semantics, 2016-09-08_euc2016, 2016-05-19_cfprotocol, 2016-07-19_curr
File Edit Selection Find View Goto Tools Project Preferences Help
sem.erl
Line 73, Column 30
Spaces: 2
Erlang

```erlang
%% Expression %% ==========================================

-type expr()    :: str() | var() | select() | cnd() | app().          % (1)
-type str()     :: {str, S::string()}.                                 % (2)
-type var()     :: {var, N::string()}.                                 % (3)
-type param()   :: {param, N::string(), Pl::boolean()}.                % (4)
-type fut()     :: {fut, R::pos_integer(), Lo::[param()]}.             % (5)
-type select()  :: {select, C::pos_integer(), U::fut()}.               % (6)
-type cnd()     :: {cnd, Xc::[expr()], Xt::[expr()], Xe::[expr()]}.    % (7)
-type app()     :: {app, C::pos_integer(), Lambda::lam() | var(),      % (8)
                    Fa::#{string() => [expr()]}}.

%% Task Signature %% ======================================

-type correl()  :: {correl, Lc::[string()]}.                           % (9)
-type inparam() :: param() | correl().                                 % (10)
-type sign()    :: {sign, Lo::[param()], Li::[inparam()]}.             % (11)

%% Lambda Term %% =========================================

-type lam()     :: {lam, S::sign(), B::body()}.                        % (12)

% Task Body
-type body()    :: natbody() | forbody().                              % (13)
-type natbody() :: {natbody, Fb::#{string() => [expr()]}}.             % (14)
-type forbody() :: {forbody, L::lang(), S::string()}.                  % (15)
-type lang()    :: bash | perl | python | r.                           % (16)

%% Evaluation Context %% ==================================

-type ctx()     :: {Rho   :: #{string() => [expr()]},                  % (17)
                    Mu    :: fun( ( app() ) -> fut() ),
                    Gamma :: #{string() => lam()},
                    Omega :: #{{N::string(), R::pos_integer()} => [expr()]}}.


%% ========================================================
%% Predicates
%% ========================================================

%% Normal Form %% =========================================

-spec pnormal( X ) -> boolean()                                        % (18)
when X :: #{string() => [expr()]} | [expr()] | expr().

pnormal( F ) when is_map( F )  -> pnormal( maps:values( F ) );        % (19)
pnormal( L ) when is_list( L ) -> lists:all( fun pnormal/1, L );      % (20,21
pnormal( {str, _} )            -> true;                                % (22)
```



4.1 Expression

The first of two items constituting Cuneiform abstract program is the query expression list $x \in [\mathsf{Expr}]$ where $[\mathsf{Expr}]$ is the set of lists whose elements are expressions. An expression $x \in \mathsf{Expr}$, in turn, is either a string literal, a variable, a future channel selection, a conditional, or a task application:

$$\mathsf{Expr} = \mathsf{Str} \cup \mathsf{Var} \cup \mathsf{Select} \cup \mathsf{Cnd} \cup \mathsf{App} \qquad (1)$$

We introduce each of these syntactical items in turn. Figure 3 summarizes the abstract syntax of Cuneiform expressions.

4.1.1 String Literal

The string literal is Cuneiform's only basic data type. In Cuneiform's abstract syntax, a string literal is represented as an abstract term of type str, i.e., a 2-tuple whose first element is the symbol **str**. The second element of the tuple is a character sequence s holding the content of the string literal. Thus, let the set String denote the set of strings (quoted character sequences), then the set Str of string literals is given by the rule:

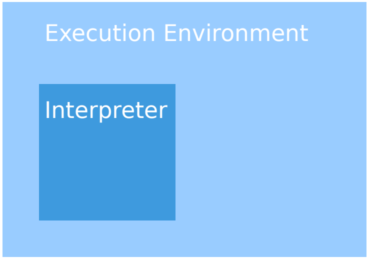
$$\frac{s \in \mathsf{String}}{(\mathbf{str}, s) \in \mathsf{Str}} \qquad (2)$$

4.1.2 Variable

Variables are placeholders for expression lists. In the abstract syntax, a variable is represented by an abstract term of type var. It comprises the symbol **var** and a character sequence holding the variable's identifier. Thus, let String denote the set of strings, then the set Var is given by the rule:

$$\frac{n \in \mathsf{String}}{(\mathbf{var}, n) \in \mathsf{Var}} \qquad (3)$$

Two Modeling Challenges (ii)



Interpreter

- Reduction of query expression

Execution Environment

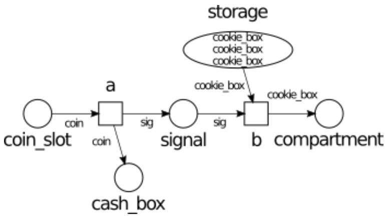
- Distributed System

How to model distributed systems?

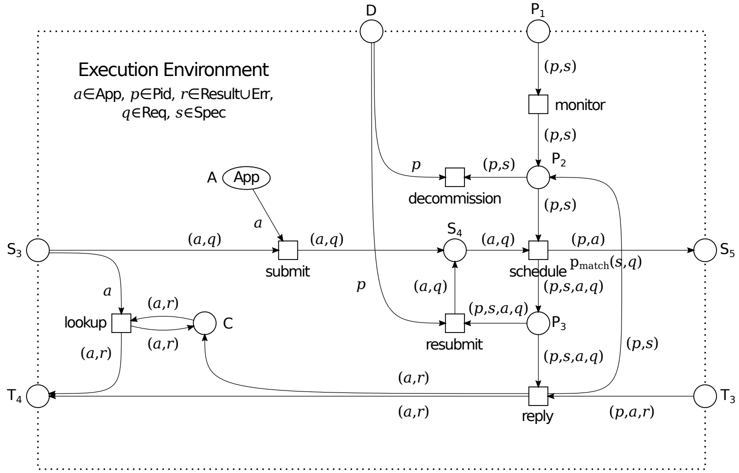
How to Model Distributed Systems

Petri Nets

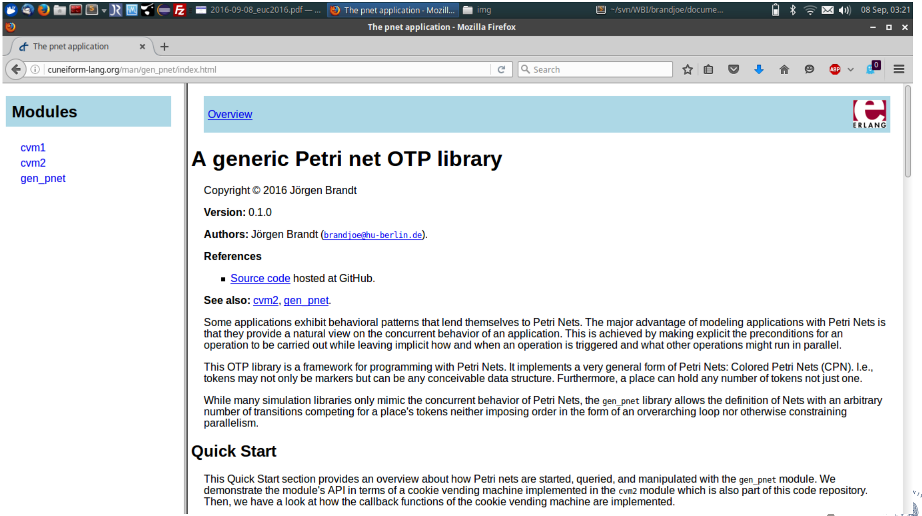
- Mature theory
  - Liveness
  - Invariants
  - Traps/Cotraps
  - . . .
- Local decision/synchronization
- Parallel execution of independent transitions



Modeling the Cuneiform Execution Environment



How to do Petri Nets in Erlang


2016-09-08_euc2016.pdf — ...
The pnet application - Mozill...
img
~/svn/WBI/brandjoe/docume...
08 Sep, 03:21
The pnet application - Mozilla Firefox
The pnet application
cuneiform-lang.org/man/gen_pnet/index.html
Search

Modules

- cvm1
- cvm2
- gen_pnet

Overview

A generic Petri net OTP library

Copyright © 2016 Jörgen Brandt

Version: 0.1.0

Authors: Jörgen Brandt (`brandjoe@hu-berlin.de`).

References

- Source code hosted at GitHub.

See also: cvm2, gen_pnet.

Some applications exhibit behavioral patterns that lend themselves to Petri Nets. The major advantage of modeling applications with Petri Nets is that they provide a natural view on the concurrent behavior of an application. This is achieved by making explicit the preconditions for an operation to be carried out while leaving implicit how and when an operation is triggered and what other operations might run in parallel.

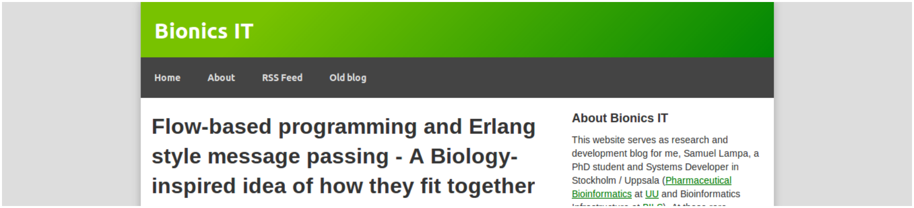
This OTP library is a framework for programming with Petri Nets. It implements a very general form of Petri Nets: Colored Petri Nets (CPN). I.e., tokens may not only be markers but can be any conceivable data structure. Furthermore, a place can hold any number of tokens not just one.

While many simulation libraries only mimic the concurrent behavior of Petri Nets, the `gen_pnet` library allows the definition of Nets with an arbitrary number of transitions competing for a place's tokens neither imposing order in the form of an orverarching loop nor otherwise constraining parallelism.

Quick Start

This Quick Start section provides an overview about how Petri nets are started, queried, and manipulated with the `gen_pnet` module. We demonstrate the module's API in terms of a cookie vending machine implemented in the `cvm2` module which is also part of this code repository. Then, we have a look at how the callback functions of the cookie vending machine are implemented.

Flow-based programming revisited



Bionics IT

Home About RSS Feed Old blog

Flow-based programming and Erlang style message passing - A Biology-inspired idea of how they fit together

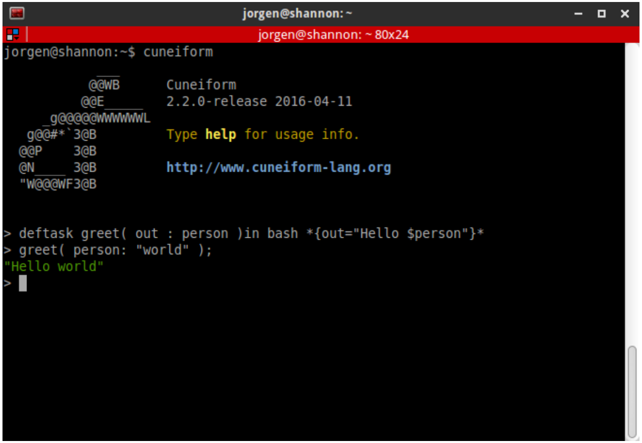
About Bionics IT

This website serves as research and development blog for me, Samuel Lampa, a PhD student and Systems Developer in Stockholm / Uppsala (Pharmaceutical Bioinformatics at UU and Bioinformatics

- System languages for heavy lifting
- Large-scale data analysis is hard
  - Because programming languages are hard
  - Because distributed systems are hard
- Erlang is good fit
  - Because FP is already close to operational semantics style notation
  - Because Erlang process model already close to Petri Nets

REPL

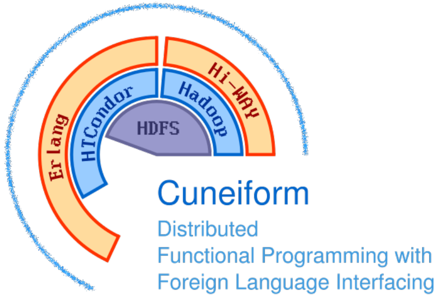
cuneiform-lang.org



```
jorgen@shannon:~$ cuneiform

          ___
          @@WB      Cuneiform
         @@E_____   2.2.0-release 2016-04-11
     _g@@@@@WWWWWWL
   g@@#*`3@B        Type help for usage info.
  @@P    3@B
  @N____ 3@B        http://www.cuneiform-lang.org
  "W@@@WF3@B


> deftask greet( out : person )in bash *{out="Hello $person"}*
> greet( person: "world" );
"Hello world"
> 
```

Conclusion

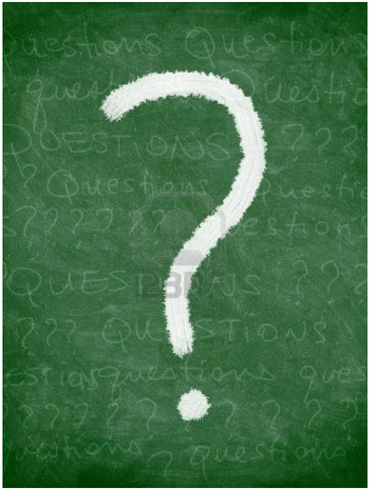
- Functional Programming + Foreign Languages
- Distributed Execution
  - Local Multicore
  - HTCondor
  - Hadoop
- Automatic Parallelization
- Expressive data analysis workflows
- Foreign Language Interface
  - Bash
  - Perl
  - ...



cuneiform-lang.org

SOAMED

Questions?



Questions?