PyPES Documentation

Release 0.99.2

Richard Bergmair

September 24, 2009

CONTENTS

1 Tutorial **3**
- 1.1 Basic Input/Output . . . 3
- 1.2 Scoping . . . 4
- 1.3 Rewriting . . . 5
- 1.4 Inferencing . . . 6

2 Library Reference **11**
- 2.1 `pypes.bin` — scripts . . . 11
- 2.2 `pypes.utils` — Utilities . . . 12
- 2.3 `pypes.proto` — ProtoForm Object Data Structure . . . 15
- 2.4 `pypes.codecs_` — Input/Output Formats for ProtoForms . . . 26
- 2.5 `pypes.rewriting` — ProtoForm Rewriting . . . 30
- 2.6 `pypes.scoping` — ProtoForm Scoping . . . 31
- 2.7 `pypes.infer` — Inference Engines & Evaluation Environment . . . 32
- 2.8 `pypes.infer.mcpiet` — The Monte Carlo Pseudo Inference Engine for Text (McPIET) . . . 34

3 Richard's Programmer Guidelines and Design Contract **37**
- 3.1 Introduction & Initial Considerations . . . 37
- 3.2 Names . . . 40
- 3.3 Memory and References . . . 42

4 Indices and tables **43**

Module Index **45**

Index **47**

Contents:

CHAPTER

ONE

TUTORIAL

1.1 Basic Input/Output

Let's start by reading an MRS from an XML file:

```
import gzip;
import codecs;
utf8reader = codecs.getreader( "utf8" );
f_ = gzip.open( "dta/test/mrs-121.mrs.xml.gz", "rb" );
f = utf8reader( f_ );
mrs121xml = f.read();
f.close();
f_.close();
```

We now have a string `mrs121xml`:

```
print( mrs121xml );
```

will give us:

```
<mrs>
<label vid='1'/><var vid='2'/>
<ep cfrom='0' cto='4'><pred>_EVERY_Q_REL</pred><label vid='3'/>
<fvpair><rargname>ARG0</rargname><var vid='4' sort='x'>
<extrapair><path>PERS</path><value>3</value></extrapair>
<extrapair><path>NUM</path><value>SG</value></extrapair>
<extrapair><path>IND</path><value>+</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='6' sort='h'></var></fvpair>
<fvpair><rargname>BODY</rargname><var vid='5' sort='h'></var></fvpair></ep>
<ep cfrom='6' cto='8'><spred>_cat_n_1_rel</spred><label vid='7'/>
[...]</ep>
[...]
<hcons hreln='qeq'><hi><var vid='12' sort='h'></var></hi><lo><var vid='13' sort='h'></var></lo></hcon
<hcons hreln='qeq'><hi><var vid='6' sort='h'></var></hi><lo><var vid='7' sort='h'></var></lo></hcons>
</mrs>
```

then:

```
from pypes.codecs_ import mrx_decode;
pf121_ = mrx_decode( mrs121xml );
```

This gives us the "lambdaified" ("frozen") representation, which we now have to "logify" ("thaw"):

```
from pypes.proto import *;
pf121 = pf121_( sig=ProtoSig() );
```

And then we can encode the result, using `PFT`, the textual representation for protoforms:

```
from pypes.codecs_ import pft_encode;

print( pft_encode( pf121, pretty=False, linebreaks=True ) );
```

will give us:

```
{  3: ? |every_q|[ PERS='3', IND='+', NUM='SG', cfrom='0', cto='4' ] x4 6 5;
   7: ? |cat_n_1|[ cto='8', cfrom='6' ]( arg0=x4 );
   8: ? |chase_v_1|[ MOOD='INDICATIVE', TENSE='PAST', PROG='-', cto='15', SF='PROP', cfrom='10', PERF
  10: ? |some_q_indiv|[ PERS='3', IND='+', NUM='SG', cfrom='17', cto='20' ] x9 12 11;
  13: ? |dog_n_1|[ cto='25', cfrom='22' ]( arg0=x9 );
      ? 12 ^ 13;
      ? 6 ^ 7 }
```

We can also do prettyprinting as follows:

```
print( pft_encode( pf121 ) );
```

results in:

```
{     |every_q| x4 2 __;
   4: |cat_n_1|( arg0=x4 );
      |chase_v_1|( arg0=e2, arg1=x4, arg2=x9 );
      |some_q_indiv| x9 3 __;
   1: |dog_n_1|( arg0=x9 );
      3 ^ 1;
      2 ^ 4 }
```

1.2 Scoping

1.2.1 Recursivization

You can also do this:

```
from pypes.scoping import Solver, Recursivizer;

with Solver( pf121 ) as solver:
  solution = solver.solve_all();
  assert solution is not None;
  with Recursivizer( solution ) as recursivizer:
    pf121_rec = recursivizer.recursivize();
    print( pft_encode( pf121_rec ) );
```

gives this:

```
{ |every_q| x4 { |cat_n_1|( arg0=x4 ) } __;
  |chase_v_1|( arg0=e2, arg1=x4, arg2=x9 );
  |some_q_indiv| x9 { |dog_n_1|( arg0=x9 ) } __ }
```

1.2.2 Enumeration

To enumerate, you do this:

```
from pypes.scoping import Enumerator;

with Solver( pf121 ) as solver:
  solution = solver.solve_all();
  with Enumerator( solution ) as enumerator:
    for solution in enumerator.enumerate():
      with Recursivizer( solution ) as recursivizer:
        pf121_enu = recursivizer.recursivize();
        print( pft_encode( pf121_enu ) );
```

This will give you:

```
{ |every_q| x4 { |cat_n_1|( arg0=x4 ) }
    { |some_q_indiv| x9 { |dog_n_1|( arg0=x9 ) }
        { |chase_v_1|( arg0=e2, arg1=x4, arg2=x9 ) } } }

{ |some_q_indiv| x9 { |dog_n_1|( arg0=x9 ) }
    { |every_q| x4 { |cat_n_1|( arg0=x4 ) }
        { |chase_v_1|( arg0=e2, arg1=x4, arg2=x9 ) } } }
```

1.3 Rewriting

1.3.1 First-Order Rewriting

In order to apply first-order rewriting, you can do this sort of thing:

```
from pypes.rewriting import erg_to_basic;

print( pft_encode( erg_to_basic( pf121 ) ) );
print( pft_encode( erg_to_basic( pf121_rec ) ) );
print( pft_encode( erg_to_basic( pf121_enu ) ) );
```

giving the following result:

```
>>> print( pft_encode( erg_to_basic( pf121 ) ) );
{     ALL x4 4 __;
   3: |cat_n_1|( arg0=x4 );
      |chase_v_1|( KEY=e2, arg1=x4, arg2=x9 );
      SOME x9 1 __;
   2: |dog_n_1|( arg0=x9 );
      1 ^ 2;
      4 ^ 3 }
```

```
>>> print( pft_encode( erg_to_basic( pf121_rec ) ) );
{ ALL x4 { |cat_n_1|( arg0=x4 ) } __;
  |chase_v_1|( KEY=e2, arg1=x4, arg2=x9 );
  SOME x9 { |dog_n_1|( arg0=x9 ) } __ }

>>> print( pft_encode( erg_to_basic( pf121_enu ) ) );
{ SOME x9 { |dog_n_1|( arg0=x9 ) }
    { ALL x4 { |cat_n_1|( arg0=x4 ) }
        { |chase_v_1|( KEY=e2, arg1=x4, arg2=x9 ) } } }
```

1.3.2 Syllogistic Forms

To get syllogistic forms:

```
from pypes.rewriting import mr_to_dsf, erg_to_bdsf;

pf121 = pf121_( sig=ProtoSig() );
print( pft_encode( mr_to_dsf( pf121 ) ) );
print( pft_encode( erg_to_bdsf( pf121 ) ) );
```

gives the following output:

```
>>> print( pft_encode( mr_to_dsf( pf121 ) ) );
{ |chase_v_1|:1( arg0=e2 );
  __ /\ __;
  { { |every_q| x4 { |cat_n_1|( arg0=x4 ) }
        { |chase_v_1|:1( arg1=x4 ) } }
    /\ { |some_q_indiv| x9 { |dog_n_1|( arg0=x9 ) }
           { |chase_v_1|:1( arg2=x9 ) } } } }

>>> print( pft_encode( erg_to_bdsf( pf121 ) ) );
{ |chase_v_1|:1( KEY=e2 );
  __ /\ __;
  { { ALL x4 { |cat_n_1|( arg0=x4 ) }
        { |chase_v_1|:1( KEY=e2, arg1=x4 ) } }
    /\ { SOME x9 { |dog_n_1|( arg0=x9 ) }
           { |chase_v_1|:1( KEY=e2, arg2=x9 ) } } } }
```

1.4 Inferencing

1.4.1 Preprocessing for Inferencing

This is how to run McPIET on the FraCaS testsuite: The process starts with the file `dta/infer/orig/fracas.bmc.xml`, which contains the test inferences produced by the FraCaS consortium in 1996 in a machine-readable format by Bill MacCartney. PyPES comes with a `Makefile` which applies edits from a `sed` script called `dta/infer/edits/fracas.bmc.xml.sed` to produce `dta/infer/edited/fracas.bmc.xml`:

```
$ make data
cp dta/infer/orig/fracas.bmc.xml dta/infer/sanitized/fracas.bmc.xml
cat dta/infer/sanitized/fracas.bmc.xml | sed -f dta/infer/edits/fracas.bmc.xml.sed > dta/infer/edited
```

Next, you need to use `src/pypes/bin/preprocess_fracas.py` to produce a couple of files in `dta/infer/fracas`:

```
$ python3 src/pypes/bin/preprocess_fracas.py
```

The files in `dta/infer/fracas` are split up by this process into various subdirectories, corresponding to the section-structure of the original fracas testsuite. For example `dta/infer/fracas/fracas-1-1` will contain all data for the 1.1 section of FraCaS. The textual data and the structures putting texts together into candidate entailments can be found in the file called `data.ts.xml`. It will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>

<testsuite>

[...]

<group>

<discourse discid="4">
  <sentence sentid="3">Every Italian man wants to be a great tenor.</sentence>
  <sentence sentid="4">Some Italian men are great tenors.</sentence>
</discourse>

<discourse discid="5">
  <sentence sentid="5">Are there Italian men who want to be a great tenor?</sentence>
</discourse>

<inference discid="6" infid="002">
  <antecedent discid="4"/>
  <consequent discid="5"/>
</inference>

</group>

[...]

<group>

<discourse discid="7">
  <sentence sentid="6">All Italian men want to be a great tenor.</sentence>
  <sentence sentid="4">Some Italian men are great tenors.</sentence>
</discourse>

[...]

</group>

[...]

</testsuite>
```

This declares discourses (with discourse ids), and the sentences contained within the discourses (with their sentence ids). If two sentences or discourses match on their cleartexts, they will also be assigned the same ids. In addition, all unique sentences and discourses will be recorded in cleartext item files in `dta/items/fracas/sentence-ctx.items` and `dta/items/fracas/discourse-ctx.items`.

In particular, the `dta/items/fracas/sentence-ctx.items` is in the input format for the [incr tsdb()]. You will have to use this tool offline to do the treebanking. The resulting `result.gz`, in the case

of FraCaS, is distributed with the ERG under `gold/fracas/result.gz`, and redistributed with PyPES under `dta/treebank/fracas.gz`. You can then read those results back into the database under `dta/items/fracas`, as follows:

```
$ python3 src/pypes/bin/read_treebank.py
  1: An Italian became the world's greatest tenor.
  2: Was there an Italian who became the world's greatest tenor?
  3: Every Italian man wants to be a great tenor.
  4: Some Italian men are great tenors.
  5: Are there Italian men who want to be a great tenor?
     [...]
639: Smith saw Jones sign the contract or cross out the crucial clause.
640: Did Smith either see Jones sign the contract or see Jones cross out the crucial clause?

--- SUMMARY ---
total number of items: 640
syntax errors: 29 (95% good)
  {261, 274, 279, 32, 296, 298, 300, 302, 305, 185, 186, 578, 323, 325, 328, 330, 333, 339, 341, 598,
scoping errors: 3 (99% good)
  {185, 186, 473}
rewriting errors: 4 (99% good)
  {32, 606, 349, 598}
total number of processed items: 611 (95% good)
```

In addition to just reading the MRS structures, this programme will also apply the rewriting machinery, to translate the MRSes into syllogistic first-order protoforms. The resulting protoforms will be stored in `dta/items/fracas/sentence-bdsf.items` and are required for the subsequent inferencing.

1.4.2 Running McPIET

Once the database under `dta/items/fracas` is populated as described before, the actual inference machineary can be run as follows:

```
$ python3 src/pypes/bin/run_testsuite.py
dta/infer/fracas/fracas-1-1
   001
   002
   [...]
   016
[...]
dta/infer/fracas/fracas-1-5
   065
   [...]
   080
```

then you can look at the results like this:

```
$ python3 src/pypes/bin/compare_annotations.py
reference file:  dta/infer/fracas/fracas-1-1/gold.tsa.xml
object file:     dta/infer/fracas/fracas-1-1/McPIETAgent.tsa.xml

      | REFERENCE               | OBJECT
------------------------------------------------------------
  001 | entailment              | entailment
  002 | entailment              | entailment
  003 | entailment              | entailment
```

```
  004 | entailment              | entailment
  005 | entailment              | entailment
  006 | contradiction           | contradiction
  007 | entailment              | entailment
  008 | entailment              | entailment
  009 | entailment              | entailment
* 010 | entailment              | unknown
  011 | entailment              | entailment
  012 | None                    | -
* 013 | entailment              | unknown
  014 | contradiction           | -
  015 | entailment              | entailment
  016 | None                    | -
-------------------------------------------------------------

error:                                    3
coverage:                                81%
acc( sys; gold ):                        84%
2w-acc( sys; gold ):                     84%
2w'-acc( sys; gold ):                   100%
acc( gold | sys = entailment ):         100%
acc( gold | sys = unknown ):              0%
acc( gold | sys = contradiction ):      100%

[...]
```

CHAPTER

TWO

LIBRARY REFERENCE

2.1 pypes.bin — scripts

pypes.bin contains main programmes and command line utilities.

sanitize_rte(*infilename, outfilename*)
is a helper script that cleans up the RTE dataset from the file named *infilename*, and writes the result to file named *outfilename*. This is normally called from the GNUmakefile.

preprocess_rte()
reads dta/infer/edited/rte*.rte.xml, and populates the output directories in dta/infer/rte with inference testing markup and the items database in dta/items/rte with cleartext items. *(not fully implemented yet)*

preprocess_rte_results()
reads the system submissions from past RTE evaluations, from dta/infer/edited/rte-results*.tar.gz, and creates annotation files in dta/infer/rte.

preprocess_fracas()
reads dta/infer/edited/fracas.bmc.xml, and populates the output directories in dta/infer/fracas with inference testing markup and the items database in dta/items/fracas with cleartext items.

extract_ergsem_smi(*ergdirname, targetdirname*)
Within the directory named *ergdirname*, this will look at the files erg.smi and core.smi to extract the grammar's SEM-I.

In the process, assertions are made, checking PyPES-internal assumptions against the grammar's SEM-I. When assertions fail, this means that the present version of the ERG is incompatible with the present version of PyPES, and adaptations have to be made on either or both sides, in order to ensure interoperability.

If the process succeeds, then the files _ergsem_smi_checker_auto.py and _erg_auto.py, which are ultimately supposed to go into src/pypes/codecs_/mrs/_smi and src/pypes/proto/lex respectively, are created in the directory named *targetdirname*.

extract_logpats()
goes through the MRS test data items, and extracts examples of predications, grouping them by the type of the MRS elementary predications, which is often useful for studying the properties of semantic representations and developing rewriting rules. The input is taken from dta/test, and the output is written to dta/pat.

read_treebank(*infilename, outdbdirname*)
reads the treebank from the file named *infilename* and populates the databases in the directory *outdbdirname* with protoforms in basic and bdsf forms; prints summary statistics on the process, and also writes those statistics to summary.txt in *outdbdirname*.

run_testsuite(*tsdirnameprefix, tsitemsdbdirname*)

runs the inference engines on a testsuite.

On the outermost level, this procedure performs an iteration over all directories which have *tsdirnameprefix* as a prefix to their name. For example, if *tsdirnameprefix* is `dta/infer/fracas/fracas-1`, then the procedure will enumerate all subdirectories of `dta/infer/fracas` which have *fracas-1* as a prefix in their name, i.e. `dta/infer/fracas/fracas-1-1`, `dta/infer/fracas/fracas-1-2`, `dta/infer/fracas/fracas-1-3`, etc.

Within each such directory, the inference testing markup file `data.ts.xml` will be consulted. Then each inference engine will be run on that testsuite, and will produce, in the same directory, an annotation file named after itself, e.g. `McPIETInferenceAgent.tsa.xml`.

Semantic inference angents will have to consult the semantic representations for sentences and discourses. These will be looked up in the items database found in the directory named *tsitemsdbdirname*.

reconsider_decisions(*tsdirnameprefix, [infile = "McPIETAgent.tsa.xml", [outfile = "McPIETAgent-reconsidered.tsa.xml"]]*)

The default strategy in McPIET is to say *entailment* when *r1* is *1.0*, contradiction when *r2* is *1.0* and *unknown* otherwise. – This leads to strict logical decisions.

This procedure can "reconsider" the entailment decisions for added robustness. The strategy used here is to decide *entailment* whenever *r1* > *r2*, and *no entailment* otherwise.

Based on *tsdirnameprefix*, this procedure goes through the various datasets, in the same way as `run_testsuite()`. It reads the *r1* and *r2* fields from the testsuite annotation in the file named *infile*, and writes the new decisions into the file named *outfile*, copying through other relevant information.

compare_decisions(*tsdirnameprefix, [objectfilename = "McPIETAgent.tsa.xml", [referencefilename = "gold.tsa.xml"]]*)

compares two annotation files containing decisions on candidate entailments, normally a set of system decisions against a gold standard.

Based on *tsdirnameprefix*, this procedure goes through the various datasets, in the same way as `run_testsuite()`. Within each directory, a file named *objectfilename* (default: `McPIETAgent.tsa.xml`) is read, and compared against a file named *referencefilename* (default: `gold.tsa.xml`).

The results are printed for quick visual inspection. In order to systematically collect statistics and export them for further analysis, use `score_decisions()` instead.

score_decisions(*prefix*)

iterates through subdirectories of the directory named by *prefix*, such as `dta/infer/fracas` or `dta/infer/rte`, and creates CSV files in `dta/infer/score` to summarize the scores for the different inference engines and subsets of the datasets.

run_unittests(*[packagename = "pypes"]*)

runs the unit tests associated with the pypes package named *packagename*. The default package name is `pypes`. If specified, the package name must be either *pypes*, or must have *pypes.* as a prefix.

2.2 `pypes.utils` — Utilities

2.2.1 `pypes.utlis.globals` — Globals

get_insttok()

get_guid()

get_runningno()

2.2.2 pypes.utils.itembank — Item Bank

class RecordManager (*(tbl, id)*)

- **sync** ()
- **id**
- **length**
- **get_ctx_str** ()
- **set_ctx_str** ()
- **reset** (*field*)
- **append_to** (*field, val*)
- **fetch_first** (*field*)
- **fetch_all** (*field*)
- **set** (*field, val*)
- **get** (*field*)

class TableManager (*(dbdirname, tblfilename)*)

- **sync** ()
- **max_id**
- **has_id** (*id*)
- **record_by_id** (*id*)
- **create_record** (*[id]*)
- **id_by_ctx_str** (*ctx_str*)
- **add_ctx_str** (*ctx_str*)

2.2.3 pypes.utils.logging_ — Extensions to Python's Logging Framework

log_attach_stderr_logger (*loggername, level*)

log_attach_file_logger (*loggername, level, logdir, prefix*)

log_critical (*sourceid, msg*)

log_error (*sourceid, msg*)

log_warn (*sourceid, msg*)

log_info (*sourceid, msg*)

log_debug_coarse (*sourceid, msg*)

log_debug (*sourceid, msg*)

log (*sourceid, level, msg*)

get_logger (*sourceid*)

print_dot ()

2.2.4 pypes.utils.os_ — Extension to Python's Miscellaneous Operating System Interfaces

listsubdirs(*pattern*)

2.2.5 pypes.utils.string_ — Miscellaneous String Handling Functions

crude_hashcode(*s*)

crude_match(*s1, s2*)

2.2.6 pypes.utils.unittest_ — Extension to Python's Unit Testing Framework

class **TestCase**()
(see also: *unittest.TestCase* in Python's standard Library)

globalstate

globalSetUp()

globalTearDown()

assertStringCrudelyEqual(*actual, expected, [msg]*)

assertStringNotCrudelyEqual(*actual, expected, [msg]*)

assertSequenceEqual(*actual, expected, [msg]*)

assertSequenceNotEqual(*actual, expected, [msg]*)

assertEquals_(*actual, expected, [msg]*)

assertNotEquals_(*actual, expected, [msg]*)

run_unittests(*[packagename]*)
is a synonym of pypes.bin.run_unittests().

2.2.7 pypes.utils.xml_ — Extension to Python's XML Framework

class **TextContentFilter**()
(see also: *xml.sax.handler.ContentHandler* in Python's standard library)

filter_textcontent(*ifile, ofile, filters, [bypass_escape=False]*)

class **XMLElementHandler**()
(see also: *xml.sax.handler.ContentHandler* in Python's standard library)

XMLELEM

class **XMLPCharElementHandler**()
(see also: *xml.sax.handler.ContentHandler* in Python's standard library)

XMLELEM

text

class **XMLProcessor**()
(see also: *xml.sax.handler.ContentHandler* in Python's standard library)

HANDLER_BYNAME

IGNORE

CHUNK_SIZE

feed(*data*)

close()

reset()

handle(*obj*)

process(*xml_*)

2.2.8 pypes.utils.mc — Metaclasses

class object_()

class Object()

class subject()

class singleton()

class kls()

2.3 pypes.proto — ProtoForm Object Data Structure

pypes.proto provides object classes for the main data structures representing protoforms and utility subjects controlling iterations and comparisons of protoforms.

2.3.1 pypes.proto.form — dynamic form objects

class Constraint(*sig, [harg, [larg]]*)

harg

larg

class SubForm(*sig*)

holes

protoforms

class Connection(*sig, [connective, [lscope, [rscope]]]*)
(see also: SubForm)

holes
(see also: SubForm.holes)

protoforms
(see also: SubForm.protoforms)

connective

lscope

rscope

class Quantification(*sig, [quantifier, [var, [rstr, [body]]]]*)
(see also: SubForm)

holes
(see also: SubForm.holes)

protoforms
(see also: SubForm.protoforms)

quantifier

var

rstr

body

class **Predication** (*sig, [predicate, [args]]*)
(see also: SubForm)

holes
(see also: SubForm.holes)

protoforms
(see also: SubForm.protoforms)

predicate

args

class **Modification** (*sig, [predicate, [args]]*)
(see also: SubForm)

holes
(see also: SubForm.holes)

protoforms
(see also: SubForm.protoforms)

modality

scope

args

class **ScopeBearer** (*sig*)

class **Handle** (*sig, [hid]*)
(see also: ScopeBearer)

hid

class **Freezer** (*sig, [content]*)
(see also: ScopeBearer)

freezelevel

content

class **ProtoForm** (*sig, [subforms, [constraints]]*)
(see also: SubForm, ScopeBearer)

holes
(see also: SubForm.holes)

protoforms
(see also: SubForm.protoforms)

roots

subforms

constraints

2.3.2 pypes.proto.sig — signature objects

class **Argument** (*predmod, [aid]*)

aid

class **ArgumentValue**(*sig*)

class **Constant**(*sig, [ident]*)
(see also: ArgumentValue)

ident

class **Sort**(*sig, [sid]*)

sid

class **Variable**(*sig, [sidvid]*)
(see also: ArgumentValue)

sort

vid

class **Functor**(*sig, [fid, [referent, [feats]]]*)

fid

referent

feats

2.3.3 pypes.proto.lex — lexical objects

class **Referent**(*sig*)

class **Word**(*sig, [lemma, [pos, [sense]]]*)
(see also: Referent)

lemma

pos

sense

class **Operator**(*sig, [otype]*)
(see also: Referent)

otype

class **Word**(*sig, [lemma, [pos, [sense]]]*)
(see also: pypes.proto.lex.basic.Word)

lemma

pos

sense

class **Operator**(*sig, [otype]*)
(see also: pypes.proto.lex.basic.Operator)

otype

2.3.4 ProtoForm Iterators

class **ProtoProcessor**()

process_constant(*inst*)

process_sort(*inst*)

process_variable(*inst*)

process_argument(*inst*)

process_word(*inst*)

process_operator(*inst*)

process_functor(*inst*)

process_predication(*inst, subform*)

process_quantification(*inst, subform*)

process_modification(*inst, subform*)

process_connection(*inst, subform*)

process_handle(*inst*)

process_scopebearer(*inst*)

process_subform(*subform*)

process_constraint(*inst*)

process_protoform(*inst, subform*)

process_constant_(*inst, ident*)

process_sort_(*inst, sid*)

process_variable_(*inst, sort, vid*)

process_argument_(*inst, aid*)

process_word_(*inst, lemma, pos, sense*)

process_operator_(*inst, otype*)

process_functor_(*inst, fid, referent, feats*)

process_predication_(*inst, subform, predicate, args*)

process_quantification_(*inst, subform, quantifier, var, rstr, body*)

process_modification_(*inst, subform, modality, args, scope*)

process_connection_(*inst, subform, connective, lscope, rscope*)

process_handle_(*inst, hid*)

process_subform_(*inst, holes, protoforms*)

process_constraint_(*inst, harg, larg*)

process_protoform_(*inst, subform, subforms, constraints*)

process(*inst*)

class **LambdaifyingProcessor**()

(see also: `ProtoProcessor`)

process_constant(*inst*)

(see: `ProtoProcessor.process_constant()`)

process_sort(*inst*)

(see: `ProtoProcessor.process_sort()`)

process_variable(*inst*)

(see: `ProtoProcessor.process_variable()`)

process_argument(*inst*)

(see: `ProtoProcessor.process_argument()`)

process_word(*inst*)

(see: `ProtoProcessor.process_word()`)

process_operator(*inst*)
(see: ProtoProcessor.process_operator())

process_functor(*inst*)
(see: ProtoProcessor.process_functor())

process_predication(*inst, subform*)
(see: ProtoProcessor.process_predication())

process_quantification(*inst, subform*)
(see: ProtoProcessor.process_quantification())

process_modification(*inst, subform*)
(see: ProtoProcessor.process_modification())

process_connection(*inst, subform*)
(see: ProtoProcessor.process_connection())

process_handle(*inst*)
(see: ProtoProcessor.process_handle())

process_freezer(*content, [freezelevel]*)

process_scopebearer(*inst*)
(see also: ProtoProcessor.process_scopebearer())

process_subform(*subform*)
(see also: ProtoProcessor.process_subform())

process_constraint(*inst*)
(see: ProtoProcessor.process_constraint())

process_protoform(*inst, subform*)
(see also: ProtoProcessor.process_protoform())

process_constant_(*inst, ident*)
(see: ProtoProcessor.process_constant_())

process_sort_(*inst, sid*)
(see: ProtoProcessor.process_sort_())

process_variable_(*inst, sort, vid*)
(see: ProtoProcessor.process_variable_())

process_argument_(*inst, aid*)
(see: ProtoProcessor.process_argument_())

process_word_(*inst, lemma, pos, sense*)
(see: ProtoProcessor.process_word_())

process_operator_(*inst, otype*)
(see: ProtoProcessor.process_operator_())

process_functor_(*inst, fid, referent, feats*)
(see: ProtoProcessor.process_functor_())

process_predication_(*inst, subform, predicate, args*)
(see: ProtoProcessor.process_predication_())

process_quantification_(*inst, subform, quantifier, var, rstr, body*)
(see: ProtoProcessor.process_quantification_())

process_modification_(*inst, subform, modality, args, scope*)
(see: ProtoProcessor.process_modification_())

process_connection_(*inst, subform, connective, lscope, rscope*)
(see: ProtoProcessor.process_connection_())

process_handle_(*inst, hid*)
(see: ProtoProcessor.process_handle_())

process_freezer_(*content, freezelevel*)

process_subform_(*inst, holes, protoforms*)
(see: ProtoProcessor.process_subform_())

process_constraint_(*inst, harg, larg*)
(see: ProtoProcessor.process_constraint_())

process_protoform_(*inst, subform, subforms, constraints*)
(see: ProtoProcessor.process_protoform_())

process(*inst*)
(see: ProtoProcessor.process())

class Lambdaifier()
(see also: LambdaifyingProcessor)

process_constant(*inst*)
(see: ProtoProcessor.process_constant())

process_sort(*inst*)
(see: ProtoProcessor.process_sort())

process_variable(*inst*)
(see: ProtoProcessor.process_variable())

process_argument(*inst*)
(see: ProtoProcessor.process_argument())

process_word(*inst*)
(see: ProtoProcessor.process_word())

process_operator(*inst*)
(see: ProtoProcessor.process_operator())

process_functor(*inst*)
(see: ProtoProcessor.process_functor())

process_predication(*inst, subform*)
(see: ProtoProcessor.process_predication())

process_quantification(*inst, subform*)
(see: ProtoProcessor.process_quantification())

process_modification(*inst, subform*)
(see: ProtoProcessor.process_modification())

process_connection(*inst, subform*)
(see: ProtoProcessor.process_connection())

process_handle(*inst*)
(see: ProtoProcessor.process_handle())

process_freezer(*content, [freezelevel]*)
(see: LambdaifyingProcessor.process_freezer())

process_scopebearer(*inst*)
(see: LambdaifyingProcessor.process_scopebearer())

process_subform(*subform*)
(see: LambdaifyingProcessor.process_subform())

process_constraint(*inst*)
(see: ProtoProcessor.process_constraint())

process_protoform(*inst, subform*)
(see: LambdaifyingProcessor.process_protoform())

process_constant_(*inst, ident*)
(see also: ProtoProcessor.process_constant_())

process_sort_(*inst, sid*)
(see also: ProtoProcessor.process_sort_())

process_variable_(*inst, sort, vid*)
(see also: ProtoProcessor.process_variable_())

process_argument_(*inst, aid*)
(see also: ProtoProcessor.process_argument_())

process_word_(*inst, lemma, pos, sense*)
(see also: ProtoProcessor.process_word_())

process_operator_(*inst, otype*)
(see also: ProtoProcessor.process_operator_())

process_functor_(*inst, fid, referent, feats*)
(see also: ProtoProcessor.process_functor_())

process_predication_(*inst, subform, predicate, args*)
(see also: ProtoProcessor.process_predication_())

process_quantification_(*inst, subform, quantifier, var, rstr, body*)
(see also: ProtoProcessor.process_quantification_())

process_modification_(*inst, subform, modality, args, scope*)
(see also: ProtoProcessor.process_modification_())

process_connection_(*inst, subform, connective, lscope, rscope*)
(see also: ProtoProcessor.process_connection_())

process_handle_(*inst, hid*)
(see also: ProtoProcessor.process_handle_())

process_freezer_(*content, freezelevel*)
(see also: LambdaifyingProcessor.process_freezer_())

process_subform_(*inst, holes, protoforms*)
(see also: ProtoProcessor.process_subform_())

process_constraint_(*inst, harg, larg*)
(see also: ProtoProcessor.process_constraint_())

process_protoform_(*inst, subform, subforms, constraints*)
(see also: ProtoProcessor.process_protoform_())

process(*inst*)
(see: ProtoProcessor.process())

lambdaify(*inst*)

lambdaify(*pf*)
is a shortcut to Lambdaifier.lambdaify().

class **BinaryProtoProcessor**()

process_constant(*inst1, inst2*)

process_sort(*inst1, inst2*)

process_variable(*inst1, inst2*)

process_argument_value(*inst1, inst2*)

process_argument(*inst1, inst2*)

process_word(*inst1, inst2*)

process_operator(*inst1, inst2*)

process_referent(*inst1, inst2*)

process_functor(*inst1, inst2*)

process_predication(*inst1, inst2, subform*)

process_quantification(*inst1, inst2, subform*)

process_modification(*inst1, inst2, subform*)

process_connection(*inst1, inst2, subform*)

process_handle(*inst1, inst2*)

process_subform(*inst1, inst2*)

process_constraint(*inst1, inst2*)

process_protoform(*inst1, inst2, subform*)

process_scopebearer(*inst1, inst2*)

process_constant_(*inst1, inst2, ident1, ident2*)

process_sort_(*inst1, inst2, sid1, sid2*)

process_variable_(*inst1, inst2, vid1, vid2, sort*)

process_argument_(*inst1, inst2, aid1, aid2*)

process_word_(*inst1, inst2, lemma1, lemma2, pos1, pos2, sense1, sense2*)

process_operator_(*inst1, inst2, otype1, otype2*)

process_functor_(*inst1, inst2, fid1, fid2, feats1, feats2, referent*)

process_predication_(*inst1, inst2, subform, predicate, args*)

process_quantification_(*inst1, inst2, subform, quantifier, var, rstr, body*)

process_modification_(*inst1, inst2, subform, modality, args, scope*)

process_connection_(*inst1, inst2, subform, connective, lscope, rscope*)

process_handle_(*inst1, inst2, hid1, hid2*)

process_subform_(*inst1, inst2, holes, protoforms*)

process_constraint_(*inst1, inst2, harg, larg*)

process_protoform_(*inst1, inst2, subform, subforms, constraints*)

process(*inst1, inst2*)

class **Comparer**()

(see also: BinaryProtoProcessor)

process_constant(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_constant())

process_sort(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_sort())

process_variable(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_variable())

process_argument_value(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_argument_value())

process_argument(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_argument())

process_word(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_word())

process_operator(*inst1, inst2*)

(see also: BinaryProtoProcessor.process_operator())

process_referent(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_referent())

process_functor(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_functor())

process_predication(*inst1, inst2, subform*)
(see also: BinaryProtoProcessor.process_predication())

process_quantification(*inst1, inst2, subform*)
(see also: BinaryProtoProcessor.process_quantification())

process_modification(*inst1, inst2, subform*)
(see also: BinaryProtoProcessor.process_modification())

process_connection(*inst1, inst2, subform*)
(see also: BinaryProtoProcessor.process_connection())

process_handle(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_handle())

process_subform(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_subform())

process_constraint(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_constraint())

process_protoform(*inst1, inst2, subform*)
(see also: BinaryProtoProcessor.process_protoform())

process_scopebearer(*inst1, inst2*)
(see also: BinaryProtoProcessor.process_scopebearer())

process_constant_(*inst1, inst2, ident1, ident2*)
(see also: BinaryProtoProcessor.process_constant_())

process_sort_(*inst1, inst2, sid1, sid2*)
(see also: BinaryProtoProcessor.process_sort_())

process_variable_(*inst1, inst2, vid1, vid2, sort*)
(see also: BinaryProtoProcessor.process_variable_())

process_argument_(*inst1, inst2, aid1, aid2*)
(see also: BinaryProtoProcessor.process_argument_())

process_word_(*inst1, inst2, lemma1, lemma2, pos1, pos2, sense1, sense2*)
(see also: BinaryProtoProcessor.process_word_())

process_operator_(*inst1, inst2, otype1, otype2*)
(see also: BinaryProtoProcessor.process_operator_())

process_functor_(*inst1, inst2, fid1, fid2, feats1, feats2, referent*)
(see also: BinaryProtoProcessor.process_functor_())

process_predication_(*inst1, inst2, subform, predicate, args*)
(see also: BinaryProtoProcessor.process_predication_())

process_quantification_(*inst1, inst2, subform, quantifier, var, rstr, body*)
(see also: BinaryProtoProcessor.process_quantification_())

process_modification_(*inst1, inst2, subform, modality, args, scope*)
(see also: BinaryProtoProcessor.process_modification_())

process_connection_(*inst1, inst2, subform, connective, lscope, rscope*)
(see also: BinaryProtoProcessor.process_connection_())

process_handle_(*inst1, inst2, hid1, hid2*)
(see also: BinaryProtoProcessor.process_handle_())

process_subform_(*inst1, inst2, holes, protoforms*)
(see also: BinaryProtoProcessor.process_subform_())

process_constraint_(*inst1, inst2, harg, larg*)
(see also: BinaryProtoProcessor.process_constraint_())

process_protoform_(*inst1, inst2, subform, subforms, constraints*)
(see also: BinaryProtoProcessor.process_protoform_())

true()

false()

meet()

join()

meet1()

join1()

meet2()

join2()

process(*inst1, inst2*)
(see: BinaryProtoProcessor.process())

pfs_leq(*pf1, pf2*)

pfs_eq(*pf1, pf2*)

pfs_leq(*pf1, pf2*)
is a shortcut to Comparer.pfs_leq().

pfs_eq(*pf1, pf2*)
is a shortcut to Comparer.pfs_eq().

class **Morpher**()
(see also: Comparer)

process_constant(*inst1, inst2*)
(see: Comparer.process_constant())

process_sort(*inst1, inst2*)
(see: Comparer.process_sort())

process_variable(*inst1, inst2*)
(see: Comparer.process_variable())

process_argument_value(*inst1, inst2*)
(see: Comparer.process_argument_value())

process_argument(*inst1, inst2*)
(see: Comparer.process_argument())

process_word(*inst1, inst2*)
(see: Comparer.process_word())

process_operator(*inst1, inst2*)
(see: Comparer.process_operator())

process_referent(*inst1, inst2*)
(see: Comparer.process_referent())

process_functor(*inst1, inst2*)
(see: Comparer.process_functor())

process_predication(*inst1, inst2, subform*)
(see: Comparer.process_predication())

process_quantification(*inst1, inst2, subform*)
(see: Comparer.process_quantification())

process_modification(*inst1, inst2, subform*)
(see: Comparer.process_modification())

process_connection(*inst1, inst2, subform*)
(see: Comparer.process_connection())

process_handle(*inst1, inst2*)
(see: Comparer.process_handle())

process_subform(*inst1, inst2*)
(see: Comparer.process_subform())

process_constraint(*inst1, inst2*)
(see: Comparer.process_constraint())

process_protoform(*inst1, inst2, subform*)
(see: Comparer.process_protoform())

process_scopebearer(*inst1, inst2*)
(see: Comparer.process_scopebearer())

process_constant_(*inst1, inst2, ident1, ident2*)
(see also: Comparer.process_constant_())

process_sort_(*inst1, inst2, sid1, sid2*)
(see also: Comparer.process_sort_())

process_variable_(*inst1, inst2, vid1, vid2, sort*)
(see also: Comparer.process_variable_())

process_argument_(*inst1, inst2, aid1, aid2*)
(see also: Comparer.process_argument_())

process_word_(*inst1, inst2, lemma1, lemma2, pos1, pos2, sense1, sense2*)
(see also: Comparer.process_word_())

process_operator_(*inst1, inst2, otype1, otype2*)
(see also: Comparer.process_operator_())

process_functor_(*inst1, inst2, fid1, fid2, feats1, feats2, referent*)
(see also: Comparer.process_functor_())

process_predication_(*inst1, inst2, subform, predicate, args*)
(see: Comparer.process_predication_())

process_quantification_(*inst1, inst2, subform, quantifier, var, rstr, body*)
(see: Comparer.process_quantification_())

process_modification_(*inst1, inst2, subform, modality, args, scope*)
(see: Comparer.process_modification_())

process_connection_(*inst1, inst2, subform, connective, lscope, rscope*)
(see: Comparer.process_connection_())

process_handle_(*inst1, inst2, hid1, hid2*)
(see also: Comparer.process_handle_())

process_subform_(*inst1, inst2, holes, protoforms*)
(see: Comparer.process_subform_())

process_constraint_(*inst1, inst2, harg, larg*)
(see: Comparer.process_constraint_())

process_protoform_(*inst1, inst2, subform, subforms, constraints*)
(see: Comparer.process_protoform_())

true()
(see also: Comparer.true())

false()
(see also: Comparer.false())

meet()
(see: Comparer.meet())

join()
(see: Comparer.join())

meet1()
(see: Comparer.meet1())

join1()
(see: Comparer.join1())

meet2()
(see also: Comparer.meet2())

join2()
(see also: Comparer.join2())

process(*inst1*, *inst2*)
(see: BinaryProtoProcessor.process())

pfs_homomorphic(*pf1*, *pf2*)

pfs_isomorphic(*pf1*, *pf2*)

pfs_homomorphic(*pf1*, *pf2*)
is a shortcut to Morpher.pfs_homomorphic().

pfs_isomorphic(*pf1*, *pf2*)
is a shortcut to Morpher.pfs_isomorphic().

2.3.5 Miscellaneous Useful Functions

sanity_check(*pf*)

analyze_as_conjunction_pf(*pf*)

recursion_check(*pf*)

sortseq(*n*)

2.4 pypes.codecs_ — Input/Output Formats for ProtoForms

pypes.codecs_ provides routines for reading from and writing to string representations of ProtoForms.

2.4.1 Decoders

class PFTDecoder(*(itemtype, lexicon)*)

decode_quoted(*pft_toks*)

decode_fid(*pft_toks*)

decode_operator(*pft_toks*)

decode_word(*pft_toks*)

decode_variable(*pft_toks*)

decode_constant(*pft_toks*)

decode_explicit_handle(*pft_toks*)

decode_anonymous_handle(*pft_toks*)

decode_features_list(*pft_toks*)

decode_arguments_list(*pft_toks*)

decode_functor(*pft_toks*)

decode_predication(*pft_toks*)

decode_freezer(*pft_toks*)

decode_quantification(*pft_toks*)

decode_modification(*pft_toks*)

decode_connection(*pft_toks*)

decode_constraint(*pft_toks*)

decode_protoform(*pft_toks*)

decode(*pft*)

pft_decode(*pf, [itemtype, [lexicon]]*)
is a shortcut to `PFTDecoder.decode()`.

class **MRXDecoder**(*[sem]*)

decode(*mrx*)

mrx_decode(*mrx, [sem]*)
is a shortcut to `MRXDecoder.decode()`.

class **MRSDecoder**(*[sem]*)

decode(*mrs*)

mrs_decode(*mrs, [sem]*)
is a shortcut to `MRSDecoder.decode()`.

2.4.2 Encoders

class **PFTEncoder**(*pf*)
(see also: `pypes.proto.LambdaifyingProcessor`, `pypes.proto.ProtoProcessor`)

process_constant(*inst*)
(see: `pypes.proto.ProtoProcessor.process_constant()`)

process_sort(*inst*)
(see: `pypes.proto.ProtoProcessor.process_sort()`)

process_variable(*inst*)
(see: `pypes.proto.ProtoProcessor.process_variable()`)

process_argument(*inst*)
(see: `pypes.proto.ProtoProcessor.process_argument()`)

process_word(*inst*)
(see: `pypes.proto.ProtoProcessor.process_word()`)

process_operator(*inst*)
(see: `pypes.proto.ProtoProcessor.process_operator()`)

process_functor(*inst*)
(see: pypes.proto.ProtoProcessor.process_functor())

process_predication(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_predication())

process_quantification(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_quantification())

process_modification(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_modification())

process_connection(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_connection())

process_handle(*inst*)
(see: pypes.proto.ProtoProcessor.process_handle())

process_freezer(*content, [freezelevel]*)
(see: pypes.proto.LambdaifyingProcessor.process_freezer())

process_scopebearer(*inst*)
(see: pypes.proto.LambdaifyingProcessor.process_scopebearer())

process_subform(*subform*)
(see: pypes.proto.LambdaifyingProcessor.process_subform())

process_constraint(*inst*)
(see: pypes.proto.ProtoProcessor.process_constraint())

process_protoform(*inst, subform*)
(see: pypes.proto.LambdaifyingProcessor.process_protoform())

process_constant_(*inst, ident*)
(see also: pypes.proto.LambdaifyingProcessor.process_constant_())

process_sort_(*inst, sid*)
(see also: pypes.proto.LambdaifyingProcessor.process_sort_())

process_variable_(*inst, sort, vid*)
(see also: pypes.proto.LambdaifyingProcessor.process_variable_())

process_argument_(*inst, aid*)
(see also: pypes.proto.LambdaifyingProcessor.process_argument_())

process_word_(*inst, lemma, pos, sense*)
(see also: pypes.proto.LambdaifyingProcessor.process_word_())

process_operator_(*inst, otype*)
(see also: pypes.proto.LambdaifyingProcessor.process_operator_())

process_functor_(*inst, fid, referent, feats*)
(see also: pypes.proto.LambdaifyingProcessor.process_functor_())

process_predication_(*inst, subform, predicate, args*)
(see also: pypes.proto.LambdaifyingProcessor.process_predication_())

process_quantification_(*inst, subform, quantifier, var, rstr, body*)
(see also: pypes.proto.LambdaifyingProcessor.process_quantification_())

process_modification_(*inst, subform, modality, args, scope*)
(see also: pypes.proto.LambdaifyingProcessor.process_modification_())

process_connection_(*inst, subform, connective, lscope, rscope*)
(see also: pypes.proto.LambdaifyingProcessor.process_connection_())

process_handle_(*inst, hid*)
(see also: pypes.proto.LambdaifyingProcessor.process_handle_())

process_freezer_(*content, freezelevel*)
(see also: pypes.proto.LambdaifyingProcessor.process_freezer_())

process_subform_(*inst, holes, protoforms*)
(see also: pypes.proto.LambdaifyingProcessor.process_subform_())

process_constraint_(*inst, harg, larg*)
(see also: pypes.proto.LambdaifyingProcessor.process_constraint_())

process_protoform_(*inst, subform, subforms, constraints*)
(see also: pypes.proto.LambdaifyingProcessor.process_protoform_())

encode(*[pretty=True, [fast_initialize=False, [linebreaks=True]]]*)

pft_encode(*pf, [pretty=True, [fast_initialize=False, [linebreaks=True]]]*)
is a shortcut to PFTEncoder.encode().

class **TreeEncoder**(*pf*)
(see also: pypes.proto.ProtoProcessor)

process_constant(*inst*)
(see: pypes.proto.ProtoProcessor.process_constant())

process_sort(*inst*)
(see: pypes.proto.ProtoProcessor.process_sort())

process_variable(*inst*)
(see: pypes.proto.ProtoProcessor.process_variable())

process_argument(*inst*)
(see: pypes.proto.ProtoProcessor.process_argument())

process_word(*inst*)
(see: pypes.proto.ProtoProcessor.process_word())

process_operator(*inst*)
(see: pypes.proto.ProtoProcessor.process_operator())

process_functor(*inst*)
(see: pypes.proto.ProtoProcessor.process_functor())

process_predication(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_predication())

process_quantification(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_quantification())

process_modification(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_modification())

process_connection(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_connection())

process_handle(*inst*)
(see: pypes.proto.ProtoProcessor.process_handle())

process_scopebearer(*inst*)
(see: pypes.proto.ProtoProcessor.process_scopebearer())

process_subform(*subform*)
(see: pypes.proto.ProtoProcessor.process_subform())

process_constraint(*inst*)
(see: pypes.proto.ProtoProcessor.process_constraint())

process_protoform(*inst, subform*)
(see: pypes.proto.ProtoProcessor.process_protoform())

process_constant_(*inst, ident*)
(see: pypes.proto.ProtoProcessor.process_constant_())

process_sort_(*inst, sid*)
(see: pypes.proto.ProtoProcessor.process_sort_())

process_variable_(*inst, sort, vid*)
(see: pypes.proto.ProtoProcessor.process_variable_())

process_argument_(*inst, aid*)
(see: pypes.proto.ProtoProcessor.process_argument_())

process_argslist(*inst, args*)

process_word_(*inst, lemma, pos, sense*)
(see also: pypes.proto.ProtoProcessor.process_word_())

process_operator_(*inst, otype*)
(see also: pypes.proto.ProtoProcessor.process_operator_())

process_functor_(*inst, fid, referent, feats*)
(see also: pypes.proto.ProtoProcessor.process_functor_())

process_predication_(*inst, subform, predicate, args*)
(see also: pypes.proto.ProtoProcessor.process_predication_())

process_quantification_(*inst, subform, quantifier, var, rstr, body*)
(see also: pypes.proto.ProtoProcessor.process_quantification_())

process_modification_(*inst, subform, modality, args, scope*)
(see also: pypes.proto.ProtoProcessor.process_modification_())

process_connection_(*inst, subform, connective, lscope, rscope*)
(see also: pypes.proto.ProtoProcessor.process_connection_())

process_handle_(*inst, hid*)
(see: pypes.proto.ProtoProcessor.process_handle_())

process_subform_(*inst, holes, protoforms*)
(see: pypes.proto.ProtoProcessor.process_subform_())

process_constraint_(*inst, harg, larg*)
(see: pypes.proto.ProtoProcessor.process_constraint_())

process_protoform_(*inst, subform, subforms, constraints*)
(see also: pypes.proto.ProtoProcessor.process_protoform_())

encode(*[utool_style=False]*)

tree_encode(*pf, [utool_style=False]*)
is a shortcut to TreeEncoder.encode().

2.5 pypes.rewriting — ProtoForm Rewriting

pypes.rewriting provides routines for rewriting protoforms.

class **Binder**(*binding*)

bind(*pf*)

bind(*binding, subform*)
is a shortcut to Binder.bind().

class **Renamer**()

process_pf(*pf*)

invert(*[rename_handles_p=True, [rename_vars_p=True, [rename_functs_p=True, [force_rename_handles_p=False]]]]*)

rename(*pf*)

rename(*pf, [rename_handles_p=True, [rename_vars_p=True, [rename_functs_p=True, [force_rename_handles_p=False]]]]*)
is a shortcut to instantiate a `Renamer`, and then call `Renamer.process_pf()`, `Renamer.invert()`, and `Renamer.rename()` on it.

class **ERGtoBasic**()

OP_Qs

WRD_Qs

OP_Cs

WRD_Cs

OP_Ms

WRD_Ms

OP_Ps

WRD_Ps

NONCTRL_WRD_Ps

rewrite(*pf*)

erg_to_basic(*pf*)
is a shortcut to `ERGtoBasic.rewrite()`.

class **CopulaResolver**()

resolve(*pf*)

resolve_copula(*pf*)
is a shortcut to `CopulaResolver.resolve()`.

class **MRtoDSF**()

rewrite(*pf*)

mr_to_dsf(*pf*)
is a shortcut to `MRtoDSF.rewrite()`.

class **Reifier**()

process_pf(*pf*)

invert()

reify(*pf*)

reify(*pf*)
is a shortcut to instantiate a `Reifier`, and then call `Reifier.process_pf()`, `Reifier.invert()`, and `Reifier.reify()` on it.

2.6 `pypes.scoping` — ProtoForm Scoping

`pypes.rewriting` provides routines for biconversion between ProtoForms and dominance constraints, and for solving and enumerating dominance constraints.

class **DomCon**()

pf

cons

cons_inv

fragments

fragments_inv

class **DomConSolution**()

domcon

chart_index

chart

cur_component

cur_root

class **Solver**(*pf*)

domcon

apply_cuts()

solve_one(*[component]*)

solve_all(*[comonent, [branching_factor]]*)

solve_one(*pf, [component]*)
is a shortcut to `Solver.solve_one()`.

solve_all(*pf, [component, [branching_factor]]*)
is a shortcut to `Solver.solve_all()`.

class **Enumerator**(*solution*)

enumerate_subcomponents(*pluggings, holes*)

enumerate_component(*component*)

enumerate()

enumerate(*solution*)
is a shortcut to `Enumerator.enumerate()`.

class **Recursivizer**(*solution*)

recursivize()

recursivize(*solution*)
is a shortcut to `Recursivizer.recursivize()`.

2.7 `pypes.infer` — Inference Engines & Evaluation Environment

`pypes.infer` provides routines for running the inference engines and using their output for evaluation purposes.

class **InferenceAgent**(*[paramid]*)
is the base class for inference agents.

paramid

reset()

process_sentence(*sentid, rec, text*)

process_discourse(*discid, rec, sents, [inf=False]*)

preprocess()

infer(*disc, antecedent, consequent*)

class YesAgent(*[paramid]*)
(see also: `InferenceAgent`)

is a baseline `InferenceAgent` class whose `infer()` method always returns `( 1.0, 0.0 )`.

class NoAgent(*[paramid]*)
(see also: `InferenceAgent`)

is a baseline `InferenceAgent` class whose `infer()` method always returns `( 0.0, 1.0 )`.

class SemanticInferenceAgent(*[paramid]*)
(see also: `InferenceAgent`)

is a subclass of `InferenceAgent` and serves as base class for inference agents which work with semantic representations from the item bank.

reset()
(see also: `InferenceAgent.reset()`)

process_sentence(*sentid, rec, text*)
(see also: `InferenceAgent.process_sentence()`)

process_discourse(*discid, rec, sents, [inf=False]*)
(see also: `InferenceAgent.process_discourse()`)

preprocess()
(see also: `InferenceAgent.preprocess()`)

infer(*disc, antecedent, consequent*)
(see also: `InferenceAgent.reset()`)

It implements `reset()`, `process_sentence()`, `process_discourse()`, and `preprocess()` to collect the appropriate protoforms from the itembank.

pfs

discs

The attributes `pfs` and `discs` are populated so that, within the `infer()` method they are available as mappings, mapping sentence ids to protoforms, and discourse ids to lists of sentence ids, respectively.

infer(*disc, antecedent, consequent*)
(see also: `InferenceAgent.infer()`)

You will need to override this to implement your own inference agent.

class TestsuiteRunner(*[(tsdirname, tsitemsdbdirname)]*)

add_agent(*agent*)

run()

run_testsuite(*tsdirnameprefix, tsitemsdbdirname*)
(see also: `pypes.bin.run_testsuite()`)

is a synonym of `pypes.bin.run_testsuite()`; controls an iteration through the directories which have *tsdirnameprefix* as a prefix. Within this iteration, it instantiates the `TestsuiteRunner`, calls `TestsuiteRunner.add_agent()` for all of the above agents, and then calls `TestsuiteRunner.run()`.

2.8 pypes.infer.mcpiet — The Monte Carlo Pseudo Inference Engine for Text (McPIET)

class LukasiewiczPropositionalLogic()

TV_MIN

TV_FALSE

TV_MAXFALSE

TV_MINTRUE

TV_TRUE

TV_MAX

tv_false()

tv_true()

tv_undesignated()

tv()

tv_is_false()

tv_is_true()

tv_is_undesignated()

tv_to_float()

p_neg(*p*)

p_strcon(*p, q*)

p_weacon(*p, q*)

p_strdis(*p, q*)

p_weadis(*p, q*)

p_imp(*p, q*)

class FirstOrderLogic(*propositional_logic*)

propositional_logic

fo_pred_open(*model, indiv_by_arg, predication*)

fo_pred_equals(*model, indiv_by_arg, predication*)

fo_quant(*model, indiv_by_var, quantification, rstr, body, var_range, outer, inner*)

fo_quant_univ(*model, indiv_by_var, quantification, rstr, body, var_range*)

fo_quant_exist(*model, indiv_by_var, quantification, rstr, body, var_range*)

fo_quant_descr(*model, indiv_by_var, quantification, rstr, body, var_range*)

class Optimizer()

optimize(*arg_range, free_args, args, function*)

class NullOptimizer()
(see also: Optimizer)

optimize(*arg_range, free_args, args, function*)
(see also: Optimizer.optimize())

class **ExhaustiveOptimizer**()
(see also: Optimizer)

optimize(*arg_range, free_args, args, function*)
(see also: Optimizer.optimize())

class **TarskiOptimizer**()
(see also: Optimizer)

optimize(*arg_range, free_args, args, function*)
(see also: Optimizer.optimize())

class **Schema**()

args

sorts

event_range

entity_range

accommodate_for_form(*pf*)

class **Model**(*schema*)

schema

matrices

class **ModelBuilder**(*logic*)

reset()

build(*schema*)

class **ModelChecker**(*logic, inner_optimizer, outer_optimizer*)

reset()

preprocess(*pfid, pf, schema*)

check(*pfid, model*)

class **McPIETAgent**(*[paramid, [propositional_logic, [firstorder_logic, [inner_optimizer, [outer_optimizer, [builder, [log2_iterations]]]]]]]*)
(see also: pypes.infer.SemanticInferenceAgent)

reset()
(see also: pypes.infer.SemanticInferenceAgent.reset())

process_sentence(*sentid, rec, text*)
(see: pypes.infer.SemanticInferenceAgent.process_sentence())

process_discourse(*discid, rec, sents, [inf=False]*)
(see: pypes.infer.SemanticInferenceAgent.process_discourse())

preprocess()
(see also: pypes.infer.SemanticInferenceAgent.preprocess())

infer(*disc, antecedent, consequent*)
(see also: pypes.infer.SemanticInferenceAgent.infer())

CHAPTER

THREE

RICHARD'S PROGRAMMER GUIDELINES AND DESIGN CONTRACT

3.1 Introduction & Initial Considerations

3.1.1 About this Document

I sometimes, completely accidentally, find myself grasping one of the deeper truths behind why a piece of code I've just written is particularly elegant or not so elegant. ...when I do, I usually cannot resist the urge to get prescriptive about it. – And this is what this document is. Here I write down a new style convention or design principle when I think of it or a new term as it gets added to the design contract which governs, but evolves together with, my software.

So, at this point, this document serves mostly as a reminder to myself, but some day perhaps also to other people who might want to contribute to my code. Hopefully, it will help me ensure that code written in the future converges towards a state of internal consistency. I do not always go back to rewrite existing code in order to enforce strict compliance with this document at all times. So, the reality of my code will usually look considerably different from the utopian vision outlined herein. Yet, within the general cycle of explorative prototyping, I do, quite frequently, throw out old code and rewrite it. In time, I hope to see some sort of evolution towards style and elegance.

These notions of style and elegance are, of course, entirely subjective, yet perfectly valid both from an aesthetic point of view and from a computer programming perspective.

The notion of the design contract, however, is intersubjective. A design contract outlines the principles that are usually present only in the form of tacit assumptions about how a particular piece of software is implemented internally, and how this relates to its external use. Most importantly, my design contract outlines how these assumptions may or may not change in future revisions, with an eye on ensuring that software modules that work together today will continue to work together tomorrow.

Obviously, there are objective truths in computer programming, but these are not within the scope of a guideline document like this.

Relation to PEP-8

A number of documents, most notably PEP-8 and the GNU Mailman Style Guide, have attained normative character for many Python projects. For my own code, I do not explicitly comply with PEP-8. Rather, I have made an effort in compiling the present style guide to ensure PEP-8 compliance, wherever there is no good reason to deviate from this well-established community standard. I explicitly state those aspects in this document, so you don't need to refer to PEP-8 directly. Wherever I deviate from PEP-8, I will make an effort to alert you to that fact and justify it.

On Bullshit

Software Engineering is, IMHO, a respectable discipline, so I find it regrettable that the language and terminology surrounding the field suffers to a great extent from bullshittery. – See the book by Harry Frankfurt for more details on this phenomenon.

In order to avoid it, I shall adopt the following strictly-no-bullshit meta rule for purposes of this document:

- Software Engineering is about how people interact with code. For the purposes of this document it shall always be preferable to refer to real people, real code, and the economics of real software engineering efforts, rather than managerial principles.

Software crises, past and present, have always been about bad people (unintelligent, lazy, uneducated, drunk, incorrectly incentivized, ...) writing bad code (long, redundant, incomprehensible, ...). It was never about the unsuccessful application of the best practices regarding the management of people or software as published in this month's Harvard Business Review or Communications of the ACM.

3.1.2 Müslifressers and Kabelbeissers

At some point in the prehistory of computer programming (let's say the late 50s or early 60s), you had to go with one of two ways of making the task of building a compiler or interpreter easy on yourself: Either you make your language be close to the machine, in which case there is a very direct correspondence between what the machine does and what the code looks like, or you make your language be based on something like lambda abstraction, in which case there is a simple and mathematically well-defined implementation for that abstraction mechanism, which then, due to its combinatorial expressiveness, automatically yields a fully-fledged programming language. In this case, the code has little or nothing to do with what the machine itself actually does. – So you could either have abstraction, at the price of being forced to abstract away from the machine, or you could be close to the machine, at the price of not having abstraction.

This lead to the formation of the two tribes of computer programmers characterized in the satirical essay Real Programmers Don't Use Pascal (also available in German). It talks about *"Real Programmers"*, whom I shall call *Kabelbeissers*, on one hand, and *Müslifressers* ("Quiche Eaters"), on the other.

Edgar Dijkstra, in his 1972 Turing Award Lecture The Humble Programmer advocated the cause of the Müslifresser. The school of thought he co-initiated holds that abstraction should be used in order to impose restrictions on the features of programming languages and their combinatorial interactions, and that programmers ought to subscribe to certain universal rules limiting what they should or shouldn't do with those languages. The idea is to limit the programmer's expressive range such that they can only think and talk about programmes in a certain way which would be conducive to overall quality and comprehensibility.

Legend has it that, at round about the same time, Seymor Cray used his machine console to type in the operating system for the CDC 7600 when it was first turned on. Here you have a man who has intimate knowledge of his machine, and who, armed with this kind of knowledge, gets the job done reliably and fuss-free; an image of expressive freedom to any true Kabelbeisser; obviously a lot more appealing than learning LISP, or whatever this week's chief Müslifresser advocates as the newspeak of the day. – Software engineering humanism.

According to Dijkstra, the "software crisis" of the late 60s happened because computer programmes started to escape the intellectual capacity of their creators. If I may offer some speculation of my own: Maybe another problem was that intellectual capacity simply went to waste. After all, the Müslifressers, on one hand, were trying to accommodate the cognitive limitations of computer programmers by making them put opening and closing backets in all the right places to surround those wonderful lambda abstractions, while the Kabelbeissers, on the other hand, thought that mastery of self-rewriting code was a particularly rewarding intellectual endeavor.

A way out was provided by languages like C++, where you have abstractions that make up a highly useful vocabulary for talking about software, while they are at the same time abstractions over what the hardware actually does. Today, you even have at your disposal multi-paradigm programming languages like Python, where data and runtime structures

have a perfectly natural and well-defined interpretation both in functional and imperative terms simultaneously. You can mix lambda abstractions and, via the Python/C interface, assembly language in the same piece of code. Dijkstra would probably have thought it impossible in 1971 that you could have your cake and eat it too.

And, where Dijkstra already lamented the introduction of hardware interrupts as an impediment to computer programming, hardware and low-level system software designs are nowadays so complex that they effectively escape the expertise of all but a very small number of specialists. Fortunately, highly optimizing compilers and virtual machines came along to save the day. And so, abstraction, nowadays, effectively facilitates, rather than impedes, the development of high-performance software.

The reason for this little history lesson is this: Technologically speaking, Müslifressers and Kabelbeissers should have gone the way of the dinosaur a long time ago. As a 21st century computer programmer, you need to be the master of both worlds: In contrast to Kabelbeissers, you should seek your expressive freedom in abstraction. You must know and effectively use all the abstraction mechanisms provided by your language. Lambda expressions are your friend! In contrast to Müslifressers, you should aspire to write high-quality software by filling your head with all those little tedious details about how the language really works under the hood. The fact that your language has garbage collection doesn't mean you can forget about memory!

There is nowadays a widespread perception that this sort of knowledge and these skills have been largely commoditized, and that computer programming is a blue-collar profession. According to this perception, the computer programmer relates to today's knowledge economy roughly as the coal worker relates to the early industrial economy.

This has lead to a proliferation of the Müslifresser, who wants to abstract away from the machine not out of technological necessity, but out of a sense of elitism and snobbery. They do not want to learn or talk too much about programming languages, operating systems, and computer hardware, for fear that their work, too, could be perceived as blue-collar. They think they can distinguish themselves by ignorance rather than competence.

I have an alternative viewpoint: Computer programming relates to the knowledge economy as literacy and numeracy relates to capitalism. In and of itself, it doesn't buy you anything, but being good at it is a prerequisite to being a player at all. It's what you do with this ability that counts, and therefore "computer programmers" aren't a commodity just because many people know how to programme any more than "writers" are just because many people are literate. Becoming a true code poet is still a worthy aspiration.

Just for the record: It should be pretty clear by now that references to Müslifressers and Kabelbeissers are meant in a derogatory sense throughout this document. Thus, a prescriptive reading should be superimposed on any descriptive statement about them.

For example, I might write this: "Kabelbeissers use the underscore convention wherever possible, as variable names only distract puny human minds from the real meaning of a variable, i.e. the one applied by the interpreter". You should read this as follows: "Keep in mind that variable names should be meaningful not only to the interpreter, but also to a human reading your code. You should therefore keep the use of the underscore convention to a minimum".

Similarly, I might write "Müslifressers have a compulsive need to state loop invariants wherever possible. Even if they need to check the termination criterion in the middle of a loop, they'd rather put up with redundant code than write the forbidden words `while True: ...`". This is to be read as "Don't be silly in shying away from `while True`. The need to check a termination criterion in the middle of a loop is perfectly common and no excuse for copy-pasting half of your codeblock".

Finally, it is regrettable that most style guides advocate the cause of the Müslifresser by telling you what you should or shouldn't do, thus limiting your expressive power. I will aim to expand your expressive power by also telling you which design principles (widely adopted by influential Müslifressers) you might *not* need to follow and by providing some inspiration for elegant designs which require some out-of-the-box thinking and are therefore seldom considered by Müslifressers.

3.1.3 Design by Contract

In this document, I will refer to someone who contributes to my code as a *developer*, not to someone who simply calls my code for developing their own programmes. The latter will be referred to as a *user*. Even when the same people

act as both users and developers for the same piece of software, it's still important that these roles not be confused.

Now, here is an applicable buzzword: *design by contract*. Users and developers are the parties to the contract, and the contract serves to regulate what they can and should expect from each other. It is perfectly possible to change the contract, as long as both parties agree to the change.

A Kabelbeisser, of course, wouldn't worry about putting this stuff in writing, as everything that needs to be said is being said in the code. – A Müslifresser, on the other hand, does all of his homework properly, by writing papers about everything, and providing termination and correctness proofs. If you are one of them: Good luck when trying to publish a proof that `open( "file" )` will open `file`.

As a practical compromise, I have found it useful to document interfaces and be relatively explicit about the design contract, but not to waste time and bore people too much by writing very verbose documentation about the actual implementation. In this document, terms of the design contract will be outlined in the numbered paragraphs.

3.1.4 Prototyping Economics

When making any kind of design decision, you will first ask yourself "What does a well-written piece of software look like?". Then you make up your mind on that question, and then you carve your code in stone. If that's the way you think about software engineering, you have already proven yourself to be a Müslifresser.

Such a static point of view is economically naive. In order to minimize risk and maximize net present value, you must ensure that a software development effort unfolds over time in such a way that as much of the functionality of the final software release be available as early as possible. Particularly for prototyping efforts in academia, science, and start-up businesses, you need to try out an idea now, and you want the fastest possible way to validate whether it works, because, more often than not, you will find that it doesn't work, and that you have to refine not just your software implementation, but the very idea itself. Any investment based on your old idea is then lost, so you want to keep it down.

At different points in time, at different points within the lifecycle of a prototyping project, you will apply quite different standards when deciding what goes for "well-written", and what doesn't. I find it quite acceptable to get a job done quick 'n' dirty first, and then, as your understanding of the problem improves, and you have increasingly verified that your general ideas work, go back and do it properly.

3.2 Names

3.2.1 Module Names, File Names, and Usage of `import`

The first thing a Müslifresser does when approaching a language like Python or Java is to mess up its namespace, both for themselves, and for everyone else who has to share the same PYTHONPATH, or CLASSPATH in Java, as them.

This leads to error conditions that are not always easy to test for. It may all work perfectly fine when you're testing and then break for mysterious reasons, usually when a user tries to install your library or, worse even, someone else's.

Therefore, it is absolutely imperative that you understand Python's naming mechanism, before you write a new package or module, or before you write a single `import` statement.

Knowing how exactly Python will resolve a given name isn't always straightforward. The exact process is documented as "platform specific".

Generally, if you import from a name like `example`, Python will look for a file called `example.py` to contain the code for module `example`. If you import from a name like `consider.an.example`, Python will look for a file called `example.py` in a directory called `consider/an`. But this lookup will be successful only if the directory `consider`, and its subdirectory `consider/an` also contain a file called `__init__.py` to mark the directory as containing a Python package.

And then the next question is *where* will the interpreter go looking for these files and directories? If the interpreter

has never imported a module of that name before, it will first look in the directory in which the script resides, i.e. that Python file which has been passed as a parameter to the interpreter, or it will look in the current working directory, if the script was read from standard input. If the file is not to be found there, it will look for it in the standard library. If it is not in the standard library either, then each directory of the PYTHONPATH will be tried in turn. As an additional complication, some users put the current working directory into their PYTHONPATH.

So when you import `foo.bar` and there are different bits of code sitting in a file `bar.py` in different directories named `foo` which contain a file `foo/__init__.py`, you are potentially in trouble.

- When you write a module, you don't necessarily know the script that has called your module, where it resides, and what other Python files are sitting in that directory.
- You don't know how the user has set their PYTHONPATH. You don't know whether it contains the current working directory, and what other libraries are listed in the PYTHONPATH before or after yours and what names are used by those libraries.

In PyPES we follow the following conventions to prevent name clashes:

Definitions:

- A *fully qualified name* is the name under which we import a module, assuming only that
  - either the module resides in a subdirectory of a directory listed as an absolute pathname in the PYTHONPATH,
  - or the module is in the Python standard library.
- *Library modules* are all modules that are accessible to Python by such a fully qualified name.
- *Scripts* are Python source files that have been called by passing their filename as a parameter to the Python interpreter, either directly, or by calling the interpreter using the `#!` directive.
- The *distribution directory* is the one single directory which is contained in the top level of the archive distributed by developers to users.

The Filesystem and the Python Namespace

1. Let the distribution directory be called `PyPES`. Users shall then ensure the `PyPES/src` and `PyPES/lib` directories are listed in their PYTHONPATH. Alternatively, they can put the contents of those directories into another directory listed in their PYTHONPATH, such as their `site-packages` or `user-packages` directories.
2. Developers shall import modules by their fully qualified names.
3. Developers shall ensure that there exists a unique name, let it by `pypes`, such that `pypes.` or `pypestest.` is a prefix of any fully qualified name of any module distributed in the `PyPES/src` directory.

This ensures that we don't mask other people's modules. – Except when some Müslifresser has decided to call something `pypes`.

Preventing Polysemy in Module Names

1. Developers shall ensure that a dot-separated string representing the fully qualified name of a PyPES module never has as a substring the prefix of another library module's fully qualified name. This includes, in particular, modules provided by the standard library and modules provided by their own code.

For example, pypes contains the module `pypes.codecs_`, rather than `pypes.codecs`. To see why this is necessary, assume we had called that module `pypes.codecs`, and consider the case where the present working directory is listed in the user's PYTHONPATH before the `PyPES/src` directory and a script is executed from inside the `PyPES/src/pypes` directory. Then `import codecs` is ambiguous between importing `pypes.codecs` and importing the standard library module `codecs`.

Another related issue is that we want to be able to do things like `from pypes import codecs`. You then have a possible name clash with the standard library module. Even if you don't need the standard library module, or have imported that module under a different name, it would be confusing to an uninitiated reader of your code that `codecs` doesn't refer to the well-known standard library module.

The Scripts Directory

1. Users shall call scripts from the dedicated `PyPES/bin` directory.
2. Developers shall ensure that neither the dedicated scripts directory `PyPES/bin` nor any other subdirectory of `PyPES` contains a file `__init__.py` either directly or in a subdirectory.
3. Developers shall ensure that the directory `PyPES/bin` never contain a file called `X.py` when `X.` is a prefix of another module's name, in particular standard library modules.

This ensures that we never mask a module whose name begins with `X.` for ourselves or other library modules. For example you can't call a script `codecs.py`, because when anybody then says `import codecs` that could refer either to that script or the standard library module.

3.2.2 Naming Conventions

Lorem ipsum.

3.3 Memory and References

Lorem ipsum.

CHAPTER

FOUR

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

P

pypes.bin, 11
pypes.codecs_, 26
pypes.infer, 32
pypes.infer.mcpiet, 34
pypes.proto, 17
pypes.proto.form, 15
pypes.proto.lex, 17
pypes.proto.lex.basic, 17
pypes.proto.lex.erg, 17
pypes.proto.sig, 16
pypes.rewriting, 30
pypes.scoping, 31
pypes.utils, 12
pypes.utils.globals, 12
pypes.utils.itembank, 13
pypes.utils.logging_, 13
pypes.utils.mc, 15
pypes.utils.os_, 14
pypes.utils.string_, 14
pypes.utils.unittest_, 14
pypes.utils.xml_, 14

INDEX

A

accommodate_for_form() (pypes.infer.mcpiet.Schema method), 35
add_agent() (pypes.infer.TestsuiteRunner method), 33
add_ctx_str() (pypes.utils.itembank.TableManager method), 13
aid (pypes.proto.sig.Argument attribute), 16
analyze_as_conjunction_pf() (in module pypes.proto), 26
append_to() (pypes.utils.itembank.RecordManager method), 13
apply_cuts() (pypes.scoping.Solver method), 32
args (pypes.infer.mcpiet.Schema attribute), 35
args (pypes.proto.form.Modification attribute), 16
args (pypes.proto.form.Predication attribute), 16
Argument (class in pypes.proto.sig), 16
ArgumentValue (class in pypes.proto.sig), 16
assertEquals_() (pypes.utils.unittest_.TestCase method), 14
assertNotEquals_() (pypes.utils.unittest_.TestCase method), 14
assertSequenceEqual() (pypes.utils.unittest_.TestCase method), 14
assertSequenceNotEqual() (pypes.utils.unittest_.TestCase method), 14
assertStringCrudelyEqual() (pypes.utils.unittest_.TestCase method), 14
assertStringNotCrudelyEqual() (pypes.utils.unittest_.TestCase method), 14

B

BinaryProtoProcessor (class in pypes.proto), 21
bind() (in module pypes.rewriting), 30
bind() (pypes.rewriting.Binder method), 30
Binder (class in pypes.rewriting), 30
body (pypes.proto.form.Quantification attribute), 16
build() (pypes.infer.mcpiet.ModelBuilder method), 35

C

chart (pypes.scoping.DomConSolution attribute), 32
chart_index (pypes.scoping.DomConSolution attribute), 32
check() (pypes.infer.mcpiet.ModelChecker method), 35
CHUNK_SIZE (pypes.utils.xml_.XMLProcessor attribute), 14
close() (pypes.utils.xml_.XMLProcessor method), 14
compare_decisions() (in module pypes.bin), 12
Comparer (class in pypes.proto), 22
Connection (class in pypes.proto.form), 15
connective (pypes.proto.form.Connection attribute), 15
cons (pypes.scoping.DomCon attribute), 32
cons_inv (pypes.scoping.DomCon attribute), 32
Constant (class in pypes.proto.sig), 17
Constraint (class in pypes.proto.form), 15
constraints (pypes.proto.form.ProtoForm attribute), 16
content (pypes.proto.form.Freezer attribute), 16
CopulaResolver (class in pypes.rewriting), 31
create_record() (pypes.utils.itembank.TableManager method), 13
crude_hashcode() (in module pypes.utils.string_), 14
crude_match() (in module pypes.utils.string_), 14
cur_component (pypes.scoping.DomConSolution attribute), 32
cur_root (pypes.scoping.DomConSolution attribute), 32

D

decode() (pypes.codecs_.MRSDecoder method), 27
decode() (pypes.codecs_.MRXDecoder method), 27
decode() (pypes.codecs_.PFTDecoder method), 27
decode_anonymous_handle() (pypes.codecs_.PFTDecoder method), 27
decode_arguments_list() (pypes.codecs_.PFTDecoder method), 27
decode_connection() (pypes.codecs_.PFTDecoder method), 27
decode_constant() (pypes.codecs_.PFTDecoder method), 27
decode_constraint() (pypes.codecs_.PFTDecoder method), 27
decode_explicit_handle() (pypes.codecs_.PFTDecoder method), 27

decode_features_list() (pypes.codecs_.PFTDecoder method), 27
decode_fid() (pypes.codecs_.PFTDecoder method), 26
decode_freezer() (pypes.codecs_.PFTDecoder method), 27
decode_functor() (pypes.codecs_.PFTDecoder method), 27
decode_modification() (pypes.codecs_.PFTDecoder method), 27
decode_operator() (pypes.codecs_.PFTDecoder method), 26
decode_predication() (pypes.codecs_.PFTDecoder method), 27
decode_protoform() (pypes.codecs_.PFTDecoder method), 27
decode_quantification() (pypes.codecs_.PFTDecoder method), 27
decode_quoted() (pypes.codecs_.PFTDecoder method), 26
decode_variable() (pypes.codecs_.PFTDecoder method), 26
decode_word() (pypes.codecs_.PFTDecoder method), 26
discs (pypes.infer.SemanticInferenceAgent attribute), 33
DomCon (class in pypes.scoping), 31
domcon (pypes.scoping.DomConSolution attribute), 32
domcon (pypes.scoping.Solver attribute), 32
DomConSolution (class in pypes.scoping), 32

E

encode() (pypes.codecs_.PFTEncoder method), 29
encode() (pypes.codecs_.TreeEncoder method), 30
entity_range (pypes.infer.mcpiet.Schema attribute), 35
enumerate() (in module pypes.scoping), 32
enumerate() (pypes.scoping.Enumerator method), 32
enumerate_component() (pypes.scoping.Enumerator method), 32
enumerate_subcomponents() (pypes.scoping.Enumerator method), 32
Enumerator (class in pypes.scoping), 32
erg_to_basic() (in module pypes.rewriting), 31
ERGtoBasic (class in pypes.rewriting), 31
event_range (pypes.infer.mcpiet.Schema attribute), 35
ExhaustiveOptimizer (class in pypes.infer.mcpiet), 34
extract_ergsem_smi() (in module pypes.bin), 11
extract_logpats() (in module pypes.bin), 11

F

false() (pypes.proto.Comparer method), 24
false() (pypes.proto.Morpher method), 26
feats (pypes.proto.sig.Functor attribute), 17
feed() (pypes.utils.xml_.XMLProcessor method), 14
fetch_all() (pypes.utils.itembank.RecordManager method), 13
fetch_first() (pypes.utils.itembank.RecordManager method), 13
fid (pypes.proto.sig.Functor attribute), 17
filter_textcontent() (pypes.utils.xml_.TextContentFilter method), 14
FirstOrderLogic (class in pypes.infer.mcpiet), 34
fo_pred_equals() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fo_pred_open() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fo_quant() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fo_quant_descr() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fo_quant_exist() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fo_quant_univ() (pypes.infer.mcpiet.FirstOrderLogic method), 34
fragments (pypes.scoping.DomCon attribute), 32
fragments_inv (pypes.scoping.DomCon attribute), 32
freezelevel (pypes.proto.form.Freezer attribute), 16
Freezer (class in pypes.proto.form), 16
Functor (class in pypes.proto.sig), 17

G

get() (pypes.utils.itembank.RecordManager method), 13
get_ctx_str() (pypes.utils.itembank.RecordManager method), 13
get_guid() (in module pypes.utils.globals), 12
get_insttok() (in module pypes.utils.globals), 12
get_logger() (in module pypes.utils.logging_), 13
get_runningno() (in module pypes.utils.globals), 12
globalSetUp() (pypes.utils.unittest_.TestCase method), 14
globalstate (pypes.utils.unittest_.TestCase attribute), 14
globalTearDown() (pypes.utils.unittest_.TestCase method), 14

H

Handle (class in pypes.proto.form), 16
handle() (pypes.utils.xml_.XMLProcessor method), 15
HANDLER_BYNAME (pypes.utils.xml_.XMLProcessor attribute), 14
harg (pypes.proto.form.Constraint attribute), 15
has_id() (pypes.utils.itembank.TableManager method), 13
hid (pypes.proto.form.Handle attribute), 16
holes (pypes.proto.form.Connection attribute), 15
holes (pypes.proto.form.Modification attribute), 16
holes (pypes.proto.form.Predication attribute), 16
holes (pypes.proto.form.ProtoForm attribute), 16
holes (pypes.proto.form.Quantification attribute), 15
holes (pypes.proto.form.SubForm attribute), 15

I

id (pypes.utils.itembank.RecordManager attribute), 13
id_by_ctx_str() (pypes.utils.itembank.TableManager method), 13
ident (pypes.proto.sig.Constant attribute), 17
IGNORE (pypes.utils.xml_.XMLProcessor attribute), 14
infer() (pypes.infer.InferenceAgent method), 33
infer() (pypes.infer.mcpiet.McPIETAgent method), 35
infer() (pypes.infer.SemanticInferenceAgent method), 33
InferenceAgent (class in pypes.infer), 32
invert() (pypes.rewriting.Reifier method), 31
invert() (pypes.rewriting.Renamer method), 30

J

join() (pypes.proto.Comparer method), 24
join() (pypes.proto.Morpher method), 26
join1() (pypes.proto.Comparer method), 24
join1() (pypes.proto.Morpher method), 26
join2() (pypes.proto.Comparer method), 24
join2() (pypes.proto.Morpher method), 26

K

kls (class in pypes.utils.mc), 15

L

Lambdaifier (class in pypes.proto), 20
lambdaify() (in module pypes.proto), 21
lambdaify() (pypes.proto.Lambdaifier method), 21
LambdaifyingProcessor (class in pypes.proto), 18
larg (pypes.proto.form.Constraint attribute), 15
lemma (pypes.proto.lex.basic.Word attribute), 17
lemma (pypes.proto.lex.erg.Word attribute), 17
length (pypes.utils.itembank.RecordManager attribute), 13
listsubdirs() (in module pypes.utils.os_), 14
log() (in module pypes.utils.logging_), 13
log_attach_file_logger() (in module pypes.utils.logging_), 13
log_attach_stderr_logger() (in module pypes.utils.logging_), 13
log_critical() (in module pypes.utils.logging_), 13
log_debug() (in module pypes.utils.logging_), 13
log_debug_coarse() (in module pypes.utils.logging_), 13
log_error() (in module pypes.utils.logging_), 13
log_info() (in module pypes.utils.logging_), 13
log_warn() (in module pypes.utils.logging_), 13
lscope (pypes.proto.form.Connection attribute), 15
LukasiewiczPropositionalLogic (class in pypes.infer.mcpiet), 34

M

matrices (pypes.infer.mcpiet.Model attribute), 35
max_id (pypes.utils.itembank.TableManager attribute), 13
McPIETAgent (class in pypes.infer.mcpiet), 35
meet() (pypes.proto.Comparer method), 24
meet() (pypes.proto.Morpher method), 26
meet1() (pypes.proto.Comparer method), 24
meet1() (pypes.proto.Morpher method), 26
meet2() (pypes.proto.Comparer method), 24
meet2() (pypes.proto.Morpher method), 26
modality (pypes.proto.form.Modification attribute), 16
Model (class in pypes.infer.mcpiet), 35
ModelBuilder (class in pypes.infer.mcpiet), 35
ModelChecker (class in pypes.infer.mcpiet), 35
Modification (class in pypes.proto.form), 16
Morpher (class in pypes.proto), 24
mr_to_dsf() (in module pypes.rewriting), 31
mrs_decode() (in module pypes.codecs_), 27
MRSDecoder (class in pypes.codecs_), 27
MRtoDSF (class in pypes.rewriting), 31
mrx_decode() (in module pypes.codecs_), 27
MRXDecoder (class in pypes.codecs_), 27

N

NoAgent (class in pypes.infer), 33
NONCTRL_WRD_Ps (pypes.rewriting.ERGtoBasic attribute), 31
NullOptimizer (class in pypes.infer.mcpiet), 34

O

Object (class in pypes.utils.mc), 15
object_ (class in pypes.utils.mc), 15
OP_Cs (pypes.rewriting.ERGtoBasic attribute), 31
OP_Ms (pypes.rewriting.ERGtoBasic attribute), 31
OP_Ps (pypes.rewriting.ERGtoBasic attribute), 31
OP_Qs (pypes.rewriting.ERGtoBasic attribute), 31
Operator (class in pypes.proto.lex.basic), 17
Operator (class in pypes.proto.lex.erg), 17
optimize() (pypes.infer.mcpiet.ExhaustiveOptimizer method), 35
optimize() (pypes.infer.mcpiet.NullOptimizer method), 34
optimize() (pypes.infer.mcpiet.Optimizer method), 34
optimize() (pypes.infer.mcpiet.TarskiOptimizer method), 35
Optimizer (class in pypes.infer.mcpiet), 34
otype (pypes.proto.lex.basic.Operator attribute), 17
otype (pypes.proto.lex.erg.Operator attribute), 17

P

p_imp() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
p_neg() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34

p_strcon() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
p_strdis() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
p_weacon() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
p_weadis() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
paramid (pypes.infer.InferenceAgent attribute), 32
pf (pypes.scoping.DomCon attribute), 32
pfs (pypes.infer.SemanticInferenceAgent attribute), 33
pfs_eq() (in module pypes.proto), 24
pfs_eq() (pypes.proto.Comparer method), 24
pfs_homomorphic() (in module pypes.proto), 26
pfs_homomorphic() (pypes.proto.Morpher method), 26
pfs_isomorphic() (in module pypes.proto), 26
pfs_isomorphic() (pypes.proto.Morpher method), 26
pfs_leq() (in module pypes.proto), 24
pfs_leq() (pypes.proto.Comparer method), 24
pft_decode() (in module pypes.codecs_), 27
pft_encode() (in module pypes.codecs_), 29
PFTDecoder (class in pypes.codecs_), 26
PFTEncoder (class in pypes.codecs_), 27
pos (pypes.proto.lex.basic.Word attribute), 17
pos (pypes.proto.lex.erg.Word attribute), 17
predicate (pypes.proto.form.Predication attribute), 16
Predication (class in pypes.proto.form), 16
preprocess() (pypes.infer.InferenceAgent method), 33
preprocess() (pypes.infer.mcpiet.McPIETAgent method), 35
preprocess() (pypes.infer.mcpiet.ModelChecker method), 35
preprocess() (pypes.infer.SemanticInferenceAgent method), 33
preprocess_fracas() (in module pypes.bin), 11
preprocess_rte() (in module pypes.bin), 11
preprocess_rte_results() (in module pypes.bin), 11
print_dot() (in module pypes.utils.logging_), 13
process() (pypes.proto.BinaryProtoProcessor method), 22
process() (pypes.proto.Comparer method), 24
process() (pypes.proto.Lambdaifier method), 21
process() (pypes.proto.LambdaifyingProcessor method), 20
process() (pypes.proto.Morpher method), 26
process() (pypes.proto.ProtoProcessor method), 18
process() (pypes.utils.xml_.XMLProcessor method), 15
process_argslist() (pypes.codecs_.TreeEncoder method), 30
process_argument() (pypes.codecs_.PFTEncoder method), 27
process_argument() (pypes.codecs_.TreeEncoder method), 29
process_argument() (pypes.proto.BinaryProtoProcessor method), 21
process_argument() (pypes.proto.Comparer method), 22
process_argument() (pypes.proto.Lambdaifier method), 20
process_argument() (pypes.proto.LambdaifyingProcessor method), 18
process_argument() (pypes.proto.Morpher method), 24
process_argument() (pypes.proto.ProtoProcessor method), 18
process_argument_() (pypes.codecs_.PFTEncoder method), 28
process_argument_() (pypes.codecs_.TreeEncoder method), 30
process_argument_() (pypes.proto.BinaryProtoProcessor method), 22
process_argument_() (pypes.proto.Comparer method), 23
process_argument_() (pypes.proto.Lambdaifier method), 21
process_argument_() (pypes.proto.LambdaifyingProcessor method), 19
process_argument_() (pypes.proto.Morpher method), 25
process_argument_() (pypes.proto.ProtoProcessor method), 18
process_argument_value() (pypes.proto.BinaryProtoProcessor method), 21
process_argument_value() (pypes.proto.Comparer method), 22
process_argument_value() (pypes.proto.Morpher method), 24
process_connection() (pypes.codecs_.PFTEncoder method), 28
process_connection() (pypes.codecs_.TreeEncoder method), 29
process_connection() (pypes.proto.BinaryProtoProcessor method), 22
process_connection() (pypes.proto.Comparer method), 23
process_connection() (pypes.proto.Lambdaifier method), 20
process_connection() (pypes.proto.LambdaifyingProcessor method), 19
process_connection() (pypes.proto.Morpher method), 25
process_connection() (pypes.proto.ProtoProcessor method), 18
process_connection_() (pypes.codecs_.PFTEncoder method), 28
process_connection_() (pypes.codecs_.TreeEncoder method), 30
process_connection_() (pypes.proto.BinaryProtoProcessor method), 22
process_connection_() (pypes.proto.Comparer method), 23
process_connection_() (pypes.proto.Lambdaifier method), 21

process_connection_() (pypes.proto.LambdaifyingProcessor method), 19
process_connection_() (pypes.proto.Morpher method), 25
process_connection_() (pypes.proto.ProtoProcessor method), 18
process_constant() (pypes.codecs_.PFTEncoder method), 27
process_constant() (pypes.codecs_.TreeEncoder method), 29
process_constant() (pypes.proto.BinaryProtoProcessor method), 21
process_constant() (pypes.proto.Comparer method), 22
process_constant() (pypes.proto.Lambdaifier method), 20
process_constant() (pypes.proto.LambdaifyingProcessor method), 18
process_constant() (pypes.proto.Morpher method), 24
process_constant() (pypes.proto.ProtoProcessor method), 17
process_constant_() (pypes.codecs_.PFTEncoder method), 28
process_constant_() (pypes.codecs_.TreeEncoder method), 29
process_constant_() (pypes.proto.BinaryProtoProcessor method), 22
process_constant_() (pypes.proto.Comparer method), 23
process_constant_() (pypes.proto.Lambdaifier method), 20
process_constant_() (pypes.proto.LambdaifyingProcessor method), 19
process_constant_() (pypes.proto.Morpher method), 25
process_constant_() (pypes.proto.ProtoProcessor method), 18
process_constraint() (pypes.codecs_.PFTEncoder method), 28
process_constraint() (pypes.codecs_.TreeEncoder method), 29
process_constraint() (pypes.proto.BinaryProtoProcessor method), 22
process_constraint() (pypes.proto.Comparer method), 23
process_constraint() (pypes.proto.Lambdaifier method), 20
process_constraint() (pypes.proto.LambdaifyingProcessor method), 19
process_constraint() (pypes.proto.Morpher method), 25
process_constraint() (pypes.proto.ProtoProcessor method), 18
process_constraint_() (pypes.codecs_.PFTEncoder method), 29
process_constraint_() (pypes.codecs_.TreeEncoder method), 30
process_constraint_() (pypes.proto.BinaryProtoProcessor method), 22
process_constraint_() (pypes.proto.Comparer method), 24
process_constraint_() (pypes.proto.Lambdaifier method), 21
process_constraint_() (pypes.proto.LambdaifyingProcessor method), 20
process_constraint_() (pypes.proto.Morpher method), 25
process_constraint_() (pypes.proto.ProtoProcessor method), 18
process_discourse() (pypes.infer.InferenceAgent method), 33
process_discourse() (pypes.infer.mcpiet.McPIETAgent method), 35
process_discourse() (pypes.infer.SemanticInferenceAgent method), 33
process_freezer() (pypes.codecs_.PFTEncoder method), 28
process_freezer() (pypes.proto.Lambdaifier method), 20
process_freezer() (pypes.proto.LambdaifyingProcessor method), 19
process_freezer_() (pypes.codecs_.PFTEncoder method), 28
process_freezer_() (pypes.proto.Lambdaifier method), 21
process_freezer_() (pypes.proto.LambdaifyingProcessor method), 20
process_functor() (pypes.codecs_.PFTEncoder method), 27
process_functor() (pypes.codecs_.TreeEncoder method), 29
process_functor() (pypes.proto.BinaryProtoProcessor method), 21
process_functor() (pypes.proto.Comparer method), 23
process_functor() (pypes.proto.Lambdaifier method), 20
process_functor() (pypes.proto.LambdaifyingProcessor method), 19
process_functor() (pypes.proto.Morpher method), 24
process_functor() (pypes.proto.ProtoProcessor method), 18
process_functor_() (pypes.codecs_.PFTEncoder method), 28
process_functor_() (pypes.codecs_.TreeEncoder method), 30
process_functor_() (pypes.proto.BinaryProtoProcessor method), 22
process_functor_() (pypes.proto.Comparer method), 23
process_functor_() (pypes.proto.Lambdaifier method), 21
process_functor_() (pypes.proto.LambdaifyingProcessor method), 19
process_functor_() (pypes.proto.Morpher method), 25
process_functor_() (pypes.proto.ProtoProcessor method), 18
process_handle() (pypes.codecs_.PFTEncoder method), 28
process_handle() (pypes.codecs_.TreeEncoder method), 29

process_handle() (pypes.proto.BinaryProtoProcessor method), 22
process_handle() (pypes.proto.Comparer method), 23
process_handle() (pypes.proto.Lambdaifier method), 20
process_handle() (pypes.proto.LambdaifyingProcessor method), 19
process_handle() (pypes.proto.Morpher method), 25
process_handle() (pypes.proto.ProtoProcessor method), 18
process_handle_() (pypes.codecs_.PFTEncoder method), 28
process_handle_() (pypes.codecs_.TreeEncoder method), 30
process_handle_() (pypes.proto.BinaryProtoProcessor method), 22
process_handle_() (pypes.proto.Comparer method), 23
process_handle_() (pypes.proto.Lambdaifier method), 21
process_handle_() (pypes.proto.LambdaifyingProcessor method), 19
process_handle_() (pypes.proto.Morpher method), 25
process_handle_() (pypes.proto.ProtoProcessor method), 18
process_modification() (pypes.codecs_.PFTEncoder method), 28
process_modification() (pypes.codecs_.TreeEncoder method), 29
process_modification() (pypes.proto.BinaryProtoProcessor method), 22
process_modification() (pypes.proto.Comparer method), 23
process_modification() (pypes.proto.Lambdaifier method), 20
process_modification() (pypes.proto.LambdaifyingProcessor method), 19
process_modification() (pypes.proto.Morpher method), 25
process_modification() (pypes.proto.ProtoProcessor method), 18
process_modification_() (pypes.codecs_.PFTEncoder method), 28
process_modification_() (pypes.codecs_.TreeEncoder method), 30
process_modification_() (pypes.proto.BinaryProtoProcessor method), 22
process_modification_() (pypes.proto.Comparer method), 23
process_modification_() (pypes.proto.Lambdaifier method), 21
process_modification_() (pypes.proto.LambdaifyingProcessor method), 19
process_modification_() (pypes.proto.Morpher method), 25
process_modification_() (pypes.proto.ProtoProcessor method), 18
process_operator() (pypes.codecs_.PFTEncoder method), 27
process_operator() (pypes.codecs_.TreeEncoder method), 29
process_operator() (pypes.proto.BinaryProtoProcessor method), 21
process_operator() (pypes.proto.Comparer method), 22
process_operator() (pypes.proto.Lambdaifier method), 20
process_operator() (pypes.proto.LambdaifyingProcessor method), 18
process_operator() (pypes.proto.Morpher method), 24
process_operator() (pypes.proto.ProtoProcessor method), 18
process_operator_() (pypes.codecs_.PFTEncoder method), 28
process_operator_() (pypes.codecs_.TreeEncoder method), 30
process_operator_() (pypes.proto.BinaryProtoProcessor method), 22
process_operator_() (pypes.proto.Comparer method), 23
process_operator_() (pypes.proto.Lambdaifier method), 21
process_operator_() (pypes.proto.LambdaifyingProcessor method), 19
process_operator_() (pypes.proto.Morpher method), 25
process_operator_() (pypes.proto.ProtoProcessor method), 18
process_pf() (pypes.rewriting.Reifier method), 31
process_pf() (pypes.rewriting.Renamer method), 30
process_predication() (pypes.codecs_.PFTEncoder method), 28
process_predication() (pypes.codecs_.TreeEncoder method), 29
process_predication() (pypes.proto.BinaryProtoProcessor method), 22
process_predication() (pypes.proto.Comparer method), 23
process_predication() (pypes.proto.Lambdaifier method), 20
process_predication() (pypes.proto.LambdaifyingProcessor method), 19
process_predication() (pypes.proto.Morpher method), 24
process_predication() (pypes.proto.ProtoProcessor method), 18
process_predication_() (pypes.codecs_.PFTEncoder method), 28
process_predication_() (pypes.codecs_.TreeEncoder method), 30
process_predication_() (pypes.proto.BinaryProtoProcessor method), 22
process_predication_() (pypes.proto.Comparer method), 23
process_predication_() (pypes.proto.Lambdaifier method), 21

process_predication_() (pypes.proto.LambdaifyingProcessor method), 19
process_predication_() (pypes.proto.Morpher method), 25
process_predication_() (pypes.proto.ProtoProcessor method), 18
process_protoform() (pypes.codecs_.PFTEncoder method), 28
process_protoform() (pypes.codecs_.TreeEncoder method), 29
process_protoform() (pypes.proto.BinaryProtoProcessor method), 22
process_protoform() (pypes.proto.Comparer method), 23
process_protoform() (pypes.proto.Lambdaifier method), 20
process_protoform() (pypes.proto.LambdaifyingProcessor method), 19
process_protoform() (pypes.proto.Morpher method), 25
process_protoform() (pypes.proto.ProtoProcessor method), 18
process_protoform_() (pypes.codecs_.PFTEncoder method), 29
process_protoform_() (pypes.codecs_.TreeEncoder method), 30
process_protoform_() (pypes.proto.BinaryProtoProcessor method), 22
process_protoform_() (pypes.proto.Comparer method), 24
process_protoform_() (pypes.proto.Lambdaifier method), 21
process_protoform_() (pypes.proto.LambdaifyingProcessor method), 20
process_protoform_() (pypes.proto.Morpher method), 25
process_protoform_() (pypes.proto.ProtoProcessor method), 18
process_quantification() (pypes.codecs_.PFTEncoder method), 28
process_quantification() (pypes.codecs_.TreeEncoder method), 29
process_quantification() (pypes.proto.BinaryProtoProcessor method), 22
process_quantification() (pypes.proto.Comparer method), 23
process_quantification() (pypes.proto.Lambdaifier method), 20
process_quantification() (pypes.proto.LambdaifyingProcessor method), 19
process_quantification() (pypes.proto.Morpher method), 24
process_quantification() (pypes.proto.ProtoProcessor method), 18
process_quantification_() (pypes.codecs_.PFTEncoder method), 28
process_quantification_() (pypes.codecs_.TreeEncoder method), 30
process_quantification_() (pypes.proto.BinaryProtoProcessor method), 22
process_quantification_() (pypes.proto.Comparer method), 23
process_quantification_() (pypes.proto.Lambdaifier method), 21
process_quantification_() (pypes.proto.LambdaifyingProcessor method), 19
process_quantification_() (pypes.proto.Morpher method), 25
process_quantification_() (pypes.proto.ProtoProcessor method), 18
process_referent() (pypes.proto.BinaryProtoProcessor method), 21
process_referent() (pypes.proto.Comparer method), 22
process_referent() (pypes.proto.Morpher method), 24
process_scopebearer() (pypes.codecs_.PFTEncoder method), 28
process_scopebearer() (pypes.codecs_.TreeEncoder method), 29
process_scopebearer() (pypes.proto.BinaryProtoProcessor method), 22
process_scopebearer() (pypes.proto.Comparer method), 23
process_scopebearer() (pypes.proto.Lambdaifier method), 20
process_scopebearer() (pypes.proto.LambdaifyingProcessor method), 19
process_scopebearer() (pypes.proto.Morpher method), 25
process_scopebearer() (pypes.proto.ProtoProcessor method), 18
process_sentence() (pypes.infer.InferenceAgent method), 33
process_sentence() (pypes.infer.mcpiet.McPIETAgent method), 35
process_sentence() (pypes.infer.SemanticInferenceAgent method), 33
process_sort() (pypes.codecs_.PFTEncoder method), 27
process_sort() (pypes.codecs_.TreeEncoder method), 29
process_sort() (pypes.proto.BinaryProtoProcessor method), 21
process_sort() (pypes.proto.Comparer method), 22
process_sort() (pypes.proto.Lambdaifier method), 20
process_sort() (pypes.proto.LambdaifyingProcessor method), 18
process_sort() (pypes.proto.Morpher method), 24
process_sort() (pypes.proto.ProtoProcessor method), 17
process_sort_() (pypes.codecs_.PFTEncoder method), 28
process_sort_() (pypes.codecs_.TreeEncoder method), 30

process_sort_() (pypes.proto.BinaryProtoProcessor method), 22
process_sort_() (pypes.proto.Comparer method), 23
process_sort_() (pypes.proto.Lambdaifier method), 21
process_sort_() (pypes.proto.LambdaifyingProcessor method), 19
process_sort_() (pypes.proto.Morpher method), 25
process_sort_() (pypes.proto.ProtoProcessor method), 18
process_subform() (pypes.codecs_.PFTEncoder method), 28
process_subform() (pypes.codecs_.TreeEncoder method), 29
process_subform() (pypes.proto.BinaryProtoProcessor method), 22
process_subform() (pypes.proto.Comparer method), 23
process_subform() (pypes.proto.Lambdaifier method), 20
process_subform() (pypes.proto.LambdaifyingProcessor method), 19
process_subform() (pypes.proto.Morpher method), 25
process_subform() (pypes.proto.ProtoProcessor method), 18
process_subform_() (pypes.codecs_.PFTEncoder method), 29
process_subform_() (pypes.codecs_.TreeEncoder method), 30
process_subform_() (pypes.proto.BinaryProtoProcessor method), 22
process_subform_() (pypes.proto.Comparer method), 23
process_subform_() (pypes.proto.Lambdaifier method), 21
process_subform_() (pypes.proto.LambdaifyingProcessor method), 20
process_subform_() (pypes.proto.Morpher method), 25
process_subform_() (pypes.proto.ProtoProcessor method), 18
process_variable() (pypes.codecs_.PFTEncoder method), 27
process_variable() (pypes.codecs_.TreeEncoder method), 29
process_variable() (pypes.proto.BinaryProtoProcessor method), 21
process_variable() (pypes.proto.Comparer method), 22
process_variable() (pypes.proto.Lambdaifier method), 20
process_variable() (pypes.proto.LambdaifyingProcessor method), 18
process_variable() (pypes.proto.Morpher method), 24
process_variable() (pypes.proto.ProtoProcessor method), 17
process_variable_() (pypes.codecs_.PFTEncoder method), 28
process_variable_() (pypes.codecs_.TreeEncoder method), 30
process_variable_() (pypes.proto.BinaryProtoProcessor method), 22
process_variable_() (pypes.proto.Comparer method), 23
process_variable_() (pypes.proto.Lambdaifier method), 21
process_variable_() (pypes.proto.LambdaifyingProcessor method), 19
process_variable_() (pypes.proto.Morpher method), 25
process_variable_() (pypes.proto.ProtoProcessor method), 18
process_word() (pypes.codecs_.PFTEncoder method), 27
process_word() (pypes.codecs_.TreeEncoder method), 29
process_word() (pypes.proto.BinaryProtoProcessor method), 21
process_word() (pypes.proto.Comparer method), 22
process_word() (pypes.proto.Lambdaifier method), 20
process_word() (pypes.proto.LambdaifyingProcessor method), 18
process_word() (pypes.proto.Morpher method), 24
process_word() (pypes.proto.ProtoProcessor method), 18
process_word_() (pypes.codecs_.PFTEncoder method), 28
process_word_() (pypes.codecs_.TreeEncoder method), 30
process_word_() (pypes.proto.BinaryProtoProcessor method), 22
process_word_() (pypes.proto.Comparer method), 23
process_word_() (pypes.proto.Lambdaifier method), 21
process_word_() (pypes.proto.LambdaifyingProcessor method), 19
process_word_() (pypes.proto.Morpher method), 25
process_word_() (pypes.proto.ProtoProcessor method), 18
propositional_logic (pypes.infer.mcpiet.FirstOrderLogic attribute), 34
ProtoForm (class in pypes.proto.form), 16
protoforms (pypes.proto.form.Connection attribute), 15
protoforms (pypes.proto.form.Modification attribute), 16
protoforms (pypes.proto.form.Predication attribute), 16
protoforms (pypes.proto.form.ProtoForm attribute), 16
protoforms (pypes.proto.form.Quantification attribute), 15
protoforms (pypes.proto.form.SubForm attribute), 15
ProtoProcessor (class in pypes.proto), 17
pypes.bin (module), 11
pypes.codecs_ (module), 26
pypes.infer (module), 32
pypes.infer.mcpiet (module), 34
pypes.proto (module), 17
pypes.proto.form (module), 15
pypes.proto.lex (module), 17
pypes.proto.lex.basic (module), 17
pypes.proto.lex.erg (module), 17
pypes.proto.sig (module), 16
pypes.rewriting (module), 30
pypes.scoping (module), 31

pypes.utils (module), 12
pypes.utils.globals (module), 12
pypes.utils.itembank (module), 13
pypes.utils.logging_ (module), 13
pypes.utils.mc (module), 15
pypes.utils.os_ (module), 14
pypes.utils.string_ (module), 14
pypes.utils.unittest_ (module), 14
pypes.utils.xml_ (module), 14

Q

Quantification (class in pypes.proto.form), 15
quantifier (pypes.proto.form.Quantification attribute), 15

R

read_treebank() (in module pypes.bin), 11
reconsider_decisions() (in module pypes.bin), 12
record_by_id() (pypes.utils.itembank.TableManager method), 13
RecordManager (class in pypes.utils.itembank), 13
recursion_check() (in module pypes.proto), 26
recursivize() (in module pypes.scoping), 32
recursivize() (pypes.scoping.Recursivizer method), 32
Recursivizer (class in pypes.scoping), 32
Referent (class in pypes.proto.lex.basic), 17
referent (pypes.proto.sig.Functor attribute), 17
Reifier (class in pypes.rewriting), 31
reify() (in module pypes.rewriting), 31
reify() (pypes.rewriting.Reifier method), 31
rename() (in module pypes.rewriting), 31
rename() (pypes.rewriting.Renamer method), 31
Renamer (class in pypes.rewriting), 30
reset() (pypes.infer.InferenceAgent method), 33
reset() (pypes.infer.mcpiet.McPIETAgent method), 35
reset() (pypes.infer.mcpiet.ModelBuilder method), 35
reset() (pypes.infer.mcpiet.ModelChecker method), 35
reset() (pypes.infer.SemanticInferenceAgent method), 33
reset() (pypes.utils.itembank.RecordManager method), 13
reset() (pypes.utils.xml_.XMLProcessor method), 15
resolve() (pypes.rewriting.CopulaResolver method), 31
resolve_copula() (in module pypes.rewriting), 31
rewrite() (pypes.rewriting.ERGtoBasic method), 31
rewrite() (pypes.rewriting.MRtoDSF method), 31
roots (pypes.proto.form.ProtoForm attribute), 16
rscope (pypes.proto.form.Connection attribute), 15
rstr (pypes.proto.form.Quantification attribute), 16
run() (pypes.infer.TestsuiteRunner method), 33
run_testsuite() (in module pypes.bin), 11
run_testsuite() (in module pypes.infer), 33
run_unittests() (in module pypes.bin), 12
run_unittests() (in module pypes.utils.unittest_), 14

S

sanitize_rte() (in module pypes.bin), 11
sanity_check() (in module pypes.proto), 26
Schema (class in pypes.infer.mcpiet), 35
schema (pypes.infer.mcpiet.Model attribute), 35
scope (pypes.proto.form.Modification attribute), 16
ScopeBearer (class in pypes.proto.form), 16
score_decisions() (in module pypes.bin), 12
SemanticInferenceAgent (class in pypes.infer), 33
sense (pypes.proto.lex.basic.Word attribute), 17
sense (pypes.proto.lex.erg.Word attribute), 17
set() (pypes.utils.itembank.RecordManager method), 13
set_ctx_str() (pypes.utils.itembank.RecordManager method), 13
sid (pypes.proto.sig.Sort attribute), 17
singleton (class in pypes.utils.mc), 15
solve_all() (in module pypes.scoping), 32
solve_all() (pypes.scoping.Solver method), 32
solve_one() (in module pypes.scoping), 32
solve_one() (pypes.scoping.Solver method), 32
Solver (class in pypes.scoping), 32
Sort (class in pypes.proto.sig), 17
sort (pypes.proto.sig.Variable attribute), 17
sorts (pypes.infer.mcpiet.Schema attribute), 35
sortseq() (in module pypes.proto), 26
SubForm (class in pypes.proto.form), 15
subforms (pypes.proto.form.ProtoForm attribute), 16
subject (class in pypes.utils.mc), 15
sync() (pypes.utils.itembank.RecordManager method), 13
sync() (pypes.utils.itembank.TableManager method), 13

T

TableManager (class in pypes.utils.itembank), 13
TarskiOptimizer (class in pypes.infer.mcpiet), 35
TestCase (class in pypes.utils.unittest_), 14
TestsuiteRunner (class in pypes.infer), 33
text (pypes.utils.xml_.XMLPCharElementHandler attribute), 14
TextContentFilter (class in pypes.utils.xml_), 14
tree_encode() (in module pypes.codecs_), 30
TreeEncoder (class in pypes.codecs_), 29
true() (pypes.proto.Comparer method), 24
true() (pypes.proto.Morpher method), 25
tv() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
TV_FALSE (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
tv_false() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
tv_is_false() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
tv_is_true() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34

tv_is_undesignated() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
TV_MAX (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
TV_MAXFALSE (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
TV_MIN (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
TV_MINTRUE (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
tv_to_float() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
TV_TRUE (pypes.infer.mcpiet.LukasiewiczPropositionalLogic attribute), 34
tv_true() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34
tv_undesignated() (pypes.infer.mcpiet.LukasiewiczPropositionalLogic method), 34

V

var (pypes.proto.form.Quantification attribute), 16
Variable (class in pypes.proto.sig), 17
vid (pypes.proto.sig.Variable attribute), 17

W

Word (class in pypes.proto.lex.basic), 17
Word (class in pypes.proto.lex.erg), 17
WRD_Cs (pypes.rewriting.ERGtoBasic attribute), 31
WRD_Ms (pypes.rewriting.ERGtoBasic attribute), 31
WRD_Ps (pypes.rewriting.ERGtoBasic attribute), 31
WRD_Qs (pypes.rewriting.ERGtoBasic attribute), 31

X

XMLELEM (pypes.utils.xml_.XMLElementHandler attribute), 14
XMLELEM (pypes.utils.xml_.XMLPCharElementHandler attribute), 14
XMLElementHandler (class in pypes.utils.xml_), 14
XMLPCharElementHandler (class in pypes.utils.xml_), 14
XMLProcessor (class in pypes.utils.xml_), 14

Y

YesAgent (class in pypes.infer), 33