

Contents

1	Module Cfg_intf : CFG - Library for the Manipulation of Context-Free Grammars	1
2	Module Cfg_impl	4
3	Module Bnf_spec	4
4	Module Bnf_pp : Pretty-printing functions for BNF-grammars	4
1	Module Cfg_intf : CFG - Library for the Manipulation of Context-Free Grammars	

```
module type SPEC =
  sig
    type t
      Terminals

    type nt
      Nonterminals

    type prod
      Productions

    type symbol =
      | NT of nt
      | T of t

    val compare_t : t -> t -> int
    val compare_nt : nt -> nt -> int
    val compare_prod : prod -> prod -> int
  end
```

Specification of grammar entities

```
module type CFG =
  sig
    module Spec :
      Cfg_intf.SPEC

    Specification of grammar elements
```

```

module TSet :
Set.S with type elt = t
module TMap :
Map.S with type key = t
module NTSet :
Set.S with type elt = nt
module NTMap :
Map.S with type key = nt
module ProdSet :
Set.S with type elt = prod * symbol list
module ProdMap :
Map.S with type key = prod * symbol list
type grammar

    The type of context-free grammars

type live_grammar

    The type of live CFGs

val empty : grammar

    empty is the empty grammar.

val add_prod :
  grammar ->
  Spec.nt -> Spec.prod -> Spec.symbol list -> grammar

    add_prod gr nt prod sl adds a production with tag prod that derives to symbol list
    sl to nonterminal nt in grammar gr.

val remove_nt : grammar -> Spec.nt -> grammar

    remove_nt gr nt removes nonterminal nt from grammar gr.

val union : grammar -> grammar -> grammar

    union gr1 gr2
    Returns the union grammar of g1 and g2.

val diff : grammar -> grammar -> grammar

    diff gr1 gr2
    Returns the difference grammar of g1 and g2.

val inter : grammar -> grammar -> grammar

```

```

inter gr1 gr2
  Returns the intersection grammar of g1 and g2.

val grammar_of_live : live_grammar -> grammar
  grammar_of_live gr converts a live grammar to a normal grammar.

val prune_unproductive : grammar -> grammar
  prune_unproductive gr prunes all unproductive entitites in gr.

val prune_nonlive : grammar -> live_grammar
  prune_nonlive gr prunes all nonlive entities in gr.

val prune_unreachable : grammar -> Spec.nt -> grammar
  prune_unreachable gr nt prunes all entities in grammar gr which cannot be reached
  from nonterminal nt.
  Raises Not_found if nt is not in gr.

val prune_unreachable_live : live_grammar -> Spec.nt -> live_grammar
  prune_unreachable_live gr nt prunes all entities in live grammar gr which cannot be
  reached from nonterminal nt. The resulting grammar contains derivation information.
  Raises Not_found if nt is not in gr.

val make_sane : grammar -> Spec.nt -> grammar
  make_sane gr nt prunes all useless entities in grammar gr using nonterminal nt as
  start symbol.
  Raises Not_found if nt is not in gr.

val make_sane_live : grammar -> Spec.nt -> live_grammar
  make_sane_live gr nt prunes all useless entities in grammar gr using nonterminal nt
  as start symbol.
  Raises Not_found if nt is not in gr.

val grammar_contents : grammar -> ProdSet.t NTMap.t
  grammar_contents gr returns a traversable representation of grammar gr.

val deriv_depth_info : live_grammar ->
  (int * int ProdMap.t) NTMap.t
  deriv_depth_info gr returns a traversable representation of live grammar gr: the left
  part of the tuple to which nonterminals are mapped tells the minimum derivation depth
  needed to completely derive the corresponding nonterminal, the right part contains a
  map of productions which are mapped to their minimum derivation depth.

```

```

val nts_in_grammar : grammar -> NTSet.t
    nts_in_grammar gr returns the set of all nonterminals in gr.

val ts_in_grammar : grammar -> TSet.t
    ts_in_grammar gr returns the set of all terminals in gr.

val prods_in_grammar : grammar -> ProdSet.t
    prods_in_grammar gr returns the set of all productions in gr.

val bounded_grammar : grammar ->
    Spec.nt -> int -> (TSet.t * grammar) list
    bounded_grammar gr nt bound computes a list of derivation levels from grammar gr,
    starting at start symbol nt and up to bound. Each level contains a set of terminals and
    a partial grammar which belong into this level.

end

Interface to context-free grammars

```

2 Module Cfg_impl

```

module Make :
    functor (Spec_ : Cfg_intf.SPEC) -> CFG with module Spec = Spec_

```

3 Module Bnf_spec

```

module Spec :
    SPEC with type t = string with type nt = string with type prod = unit
module Bnf :
    CFG with module Spec = Spec

```

4 Module Bnf_pp : Pretty-printing functions for BNF-grammars

```

val pp_prod : Format.formatter -> Bnf_spec.Bnf.Spec.symbol list -> unit
    pp_prod ppf syms prettyprint symbols list syms using prettyprinter ppf.

val pp_live_prods : Format.formatter -> int Bnf_spec.Bnf.ProdMap.t -> unit
    pp_live_prods ppf syms prettyprint live production map pm using prettyprinter ppf.

```

```

val pp_nt : Format.formatter -> string -> Bnf_spec.Bnf.ProdSet.t -> unit
    pp_nt ppf nt ps prettyprint nonterminal nt and its production set ps using prettyprinter
    ppf.

val pp_live_nt :
    Format.formatter -> string -> int * int Bnf_spec.Bnf.ProdMap.t -> unit
    pp_nt ppf nt di prettyprint live nonterminal nt and its derivation information di using
    prettyprinter ppf.

val pp_nt_map :
    Format.formatter -> Bnf_spec.Bnf.ProdSet.t Bnf_spec.Bnf.NTMap.t -> unit
    pp_nt_map ppf nts prettyprint map of nonterminals nts using prettyprinter ppf.

val pp_live_nts :
    Format.formatter ->
    (int * int Bnf_spec.Bnf.ProdMap.t) Bnf_spec.Bnf.NTMap.t -> unit
    pp_live_nts ppf nt_di prettyprint map of nonterminal derivation information nt_di using
    prettyprinter ppf.

val pp_ts : Format.formatter -> Bnf_spec.Bnf.TSet.t -> unit
    pp_ts ppf ts prettyprint set of terminals ts using prettyprinter ppf.

val pp_nts : Format.formatter -> Bnf_spec.Bnf.NTSet.t -> unit
    pp_nts ppf nts prettyprint set of nonterminals nts using prettyprinter ppf.

val pp_prods : Format.formatter -> Bnf_spec.Bnf.ProdSet.t -> unit
    pp_prods ppf prods prettyprint set of productions prods using prettyprinter ppf.

```