A Quick-Start Guide to Bedwyr v1.4

Quentin Heath
Inria Saclay–Île-de-France & LIX, École polytechnique

June 20, 2015

1 Overview

Bedwyr is a generalization of logic programming that allows model checking directly on syntactic expression.

The logic manipulated, a subset of LINC (for *lambda, induction, nabla, co-induction*), contains object-level λ-abstractions, ∇ (nominal) quantification, and (co-)inductive definitions. While LINC is a extension of higher-order intuitionistic logic, Bedwyr formulae are restricted to a fragment where connectives on the left of an implication must have invertible rules (i.e. no universal quantifier nor implication – this enables the closed-world assumption when identifying finite success and finite failure), while equalities are restricted to the L_λ fragment (allowing for the use of higher-order pattern unification).

The OCaml implementation is part of an open source project. The web page offers multiple ways to get it:

- SVN repository
- tarballs
- precompiled binaries
- GNU/Linux packages
- Windows installer

The documentation includes this quick-start guide, a reference manual and the source-code documentation.

2 Input

Input sources can be plain text definition files (`*.def`), the REPL (read-eval-print loop), the `read` predicate, and arguments to the `-d`, `-e` and `-c` CLI (command-line interface) options. Three modes are available:

definition mode is used in `*.def` files and with `-d`
toplevel mode is used in the REPL and with `-e`
term mode is used at run-time via `read` and with `-c`

	*.def	REPL	`read`	CLI
2.1 `Define p : prop.`	✓			`-d`
2.2 `X = 42.`		✓		`-e`
2.3 `#env.`	✓	✓		`-d/-e`
2.4 `1 :: 2 :: nil.`			✓	`-c`

2.1 Commands

Commands are used in definition mode to declare new types and constants, declare and define predicates, and define theorems (which are not proved, but are used as lemmas).

2.2 Queries

Queries are plain formulae (terms of type `prop`), entered at the toplevel, that Bedwyr attempts to solve. If possible, it outputs the list of solutions (substitutions of the free variables). Otherwise, if the formula is not handled by the prover (non-invertible connective on the left) or by the unifier (not L_λ), resolution aborts.

2.3 Meta-commands

Meta-commands are used both in definition mode and at the toplevel, mostly to improve the user experience by executing strictly non-logical tasks, such as input (`#include "inc.def".`), output (`#typeof X Y :: nil.`) or testing (`#assert true.`). A few of them change proof structure, but not provability (`#freezing 4.`, `#saturation 2.`).

2.4 Terms

Term-mode is a way to improve interactivity.

Listing 1: maxa.def

```
Kind ch type.

Type z  ch.
Type s  ch -> ch.

% The predicate a holds for 3, 5, and 2.
Define a : ch -> prop by
  a (s (s (s z))) ;
  a (s (s (s (s (s z))))) ;
  a (s (s z)).

% The less-than-or-equal relation
Define inductive leq : ch -> ch -> prop
by leq z _
;  leq (s N) (s M) := leq N M.

% Compute the maximum of a
Define maxa : ch -> prop
by maxa N :=
     a N /\ forall x, a x -> leq x N.
```

Listing 2: Interactive session

```
?= #env.
*** Types ***
   option : * -> *
   list : * -> *
   ch : *
*** Constants ***
   (::) : A -> list A -> list A
   nil : list A
   opnone : option A
   opsome : A -> option A
   s : ch -> ch
   z : ch
*** Predicates ***
   a : ch -> prop
 I leq : ch -> ch -> prop
   maxa : ch -> prop
   member : A -> list A -> prop
?= read Y /\ leq X Y.
 ?> s z.
Solution found:
 X = z
 Y = s z
More [y] ? y
Solution found:
 X = s z
 Y = s z
More [y] ? y
No more solutions.
?= maxa X.
Solution found:
 X = s (s (s (s (s z))))
More [y] ? y
No more solutions.
?= #exit.
```

3 Sample definition file

Listing 1 shows a complete sample definition file, with the declarations for a type and two constants, along with a few predicates.

The predicate `a` is a typical example of what must be done to build a Bedwyr example: even with a theoretically infinite search space (here, Church numerals), Bedwyr only does finite reasoning, and hence must be given an explicit description of its finite actual search space.

The use of the `inductive` keyword has two consequences. Firstly, memoization is used on the corresponding predicate; secondly, it has an impact on the way loops in computation are handled. Since the `leq` predicate obviously cannot loop, only the first aspect is used here (meaning we might as well have used the `coinductive` keyword instead).

4 REPL demo

Listing 2 shows an example of use of the interactive toplevel following the invocation of `bedwyr maxa.def`.

The `#env.` meta-command shows all declared objects (including the standard pre-declared list-related objects), and informs `leq` is inductive. The call to the queries `leq X (s z).` and `maxa X.` offer to display all solutions, one by one. A subsequent call to `#show_table leq.` would show the table filled by the second query with `leq`-headed atoms, either marked as proved or disproved.