News in Bedwyr

the 1.3 release and some future plans

Quentin Heath

INRIA-Saclay & Lix, École polytechnique

Certificates and Computation workshop, ITU Copenhagen
March 2012

Outline

Presentation

System description

Examples

Future developments

Outline

Presentation

System description

Examples

Future developments

Proof-search engine

0/1 (LINC: λ, induction, ∇, co-induction)

Proof-search engine

0/1 (LINC: λ, induction, ∇, co-induction)

- fixed points through definitions

Proof-search engine

0/1 (LINC: λ, induction, ∇, co-induction)

- fixed points through definitions
- HOAS (λ-tree), ∇ quantifier

Release

- 1.0 (doc, examples, tabling)

Release

- 1.0 (doc, examples, tabling), 1.1, 1.2

Release

- 1.0 (doc, examples, tabling), 1.1, 1.2
- 1.3 (types, syntax)

Release

- 1.0 (doc, examples, tabling), 1.1, 1.2
- 1.3 (types, syntax)

- 2.0 (Abella-compatible)

Release

- 1.0 (doc, examples, tabling), 1.1, 1.2
- 1.3 (types, syntax)
- 1.4 (improved tabling)
- ...
- 2.0 (Abella-compatible)

Release

- 1.0 (doc, examples, tabling), 1.1, 1.2
- 1.3 (types, syntax)
- 1.4 (improved tabling)
- . . .
- 2.0 (Abella-compatible)

Release

bedwyr-1.3-rc2 (ndcore & bedwyr – not tac!)
`http://slimmer.gforge.inria.fr/bedwyr/`

- doc, including ocamldoc
- building flow (tests, installation, etc)
- tarball/deb

Outline

Presentation

System description

Examples

Future developments

Syntax

- λProlog:

```
edge X Y :- arrow X Y ; arrow Y X.
path X Y :- edge X Y.
path X Y :- edge X Z , path Z Y.
arrow X Y :- ...
```

- Abella:

```
Define arrow ...
Define edge : nat -> nat -> prop by
  edge X Y := arrow X Y \/ arrow Y X.
Define path : nat -> nat -> prop by
  path X Y = edge X Y ;
  path X Y = edge X Z /\ path Z Y.
```

Syntax

`(pi f)` allows `(pi true)` or even `(f pi)`...

Syntax

`(pi f)` allows `(pi true)` or even `(f pi)`...

- λProlog:

  ```
  pi x\ pi y\ sigma z\ x => y
  ```

- Abella:

  ```
  forall x y, exists z, x -> y
  ```

Syntax

And also: type annotations for bound variables, multiline comments, better quoted strings...

```
?= (x : string\ x) =
/* this equality
 * doesn't typecheck */
   (x : nat\ x).
At line 7, characters 5-14:
Typing error: this expression has type nat -> ?40 but is used as
string -> string.
?= printstr "This is \
a
multiline\ string.\n" /\ false.
This is a
multiline string.
No.
?=
```

Type system

- simple types, no polymorphism yet: `Kind t1,t2 type.`
- parse-time checking: typed pre-terms and untyped engine
- new meta-commands (`#env`, `#typeof`)

Type system

- simple types, no polymorphism yet: `Kind t1,t2 type.`
- parse-time checking: typed pre-terms and untyped engine
- new meta-commands (`#env`, `#typeof`)

```
$ src/bedwyr examples/graph.def -e "#env."
[...]
?= #typeof F (print_arrow _) (print X) = 42.
(F (print_arrow _) (print X)) = 42 : prop
   F : (nat -> prop) -> prop -> nat
   _ : nat
   X : ?141
```

SPEC

- I/O (`fopen_out`, `fprint`)
- non-logical (`_distinct`, `_eqvt`)

SPEC

- I/O (`fopen_out`, `fprint`)
- non-logical (`_distinct`, `_eqvt`)

```
?= true \/ true.
Yes.
More [y] ?
Yes.
More [y] ?
No more solutions.
?= _distinct (true \/ true).
Yes.
More [y] ?
No more solutions.
```

SPEC

- I/O (`fopen_out`, `fprint`)
- non-logical (`_distinct`, `_eqvt`)

```
?= forall f, nabla x y, _eqvt (f x y) (f y x).
Yes.
More [y] ? y
No more solutions.
?= forall f, nabla x y, _eqvt (f x x) (f y y).
Yes.
More [y] ? y
No more solutions.
?= forall f, nabla x y, _eqvt (f x x) (f x y).
No.
```

Tabling

A small example (`graph.def`):

```
%       3
%      / \
% 1 - 2   5 - 6
%      \ /
%       4
Define arrow : nat -> nat -> prop by
  arrow 1 2; arrow 2 3;
  arrow 2 4; arrow 3 5;
  arrow 4 5; arrow 5 6.

Define edge : nat -> nat -> prop by
  edge X Y := arrow X Y \/ arrow Y X.
```

Tabling

A small example (graph.def):

```
Define edgepath : nat -> nat -> prop by
  edgepath X Y := edge X Y;
  edgepath X Y := edge X Z /\ edgepath Z Y.
```

```
$ src/bedwyr examples/graph.def
?= edgepath 1 6.
More [y] ?
Yes.
More [y] ?
[...]
```

Tabling

A small example (`graph.def`):

```
Define inductive edgepath : nat -> nat -> prop by
  edgepath X Y := edge X Y;
  edgepath X Y := edge X Z /\ edgepath Z Y.
```

```
$ src/bedwyr examples/graph.def
?= edgepath 1 6.
Yes.
More [y] ?
No more solutions.
```

Tabling

A small example (`graph.def`):

```
Define inductive edgepath : nat -> nat -> prop by
  edgepath X Y := edge X Y;
  edgepath X Y := edge X Z /\ edgepath Z Y.
```

```
$ src/bedwyr examples/graph.def
?= edgepath 1 X /\ print X /\ false.
23415263456345634563456345634563456345634563456345634563
456345634563456345634563456345634563456345634563456345[...]
```

Since `(edgepath 1 X)` is not tabled, backchaining never kicks in!

Tabling

A better example (more or less `graph-alt.def`):

```
Define inductive edgepath : nat -> nat -> prop by
  edgepath X X := node X;
  edgepath X Y := edge X Z /\ edgepath Z Y.

$ src/bedwyr examples/graph-alt.def
?= forall X Y, node X -> node Y ->
   (edgepath X Y /\ print X /\ print Y).
6665646362615655545352514645444342413635343332312625
24232221161514131211Yes.
More [y] ?
No more solutions.
```

Outline

Presentation

System description

Examples

Future developments

peterson.def

```
$ src/bedwyr examples/peterson.def -t \
  -e "#show_table path." -I
```

tictactoe.def

```
$ src/bedwyr examples/tictactoe.def -t \
  -e "#show_table notxwins." -I
```

Outline

Presentation

System description

Examples

Future developments

Future developments

- minimal vs nominal nabla

Future developments

- minimal vs nominal nabla
- Abella

Future developments: adapting Abella

- Abella:

```
Define a : nat -> prop by ...
CoDefine
  b : nat -> prop
  c : nat -> prop
by ...
```

- Bedwyr:

```
Define inductive a : nat -> prop by ...
Define
  coinductive b : nat -> prop
  c : nat -> prop
by ...
```

Future developments: adapting Abella

- Abella:

```
Define a : nat -> prop by ...
CoDefine
  b : nat -> prop
  c : nat -> prop
by ...
```

- Bedwyr:

```
Define inductive a : nat -> prop by a X := a X.
Define
  coinductive b : nat -> prop
  c : nat -> prop
by
  b X := b X ;
  c X := c X.
```

Future developments

- minimal vs nominal nabla
- Abella

Future developments

- minimal vs nominal nabla
- Abella
- tables

Future developments: tabling

- free variables
- forward chaining
- backchaining

Future developments

- minimal vs nominal nabla
- Abella
- tables

Future developments

- minimal vs nominal nabla
- Abella
- tables
- runtime errors (higher order pattern unification)

Future developments: runtime errors

```
Kind tm, fm   type.
Type all   (tm -> fm) -> fm.
Type p        tm -> fm.
Type a        tm.
Define instan : fm -> tm -> fm -> prop by
   instan (B T) T (all B).

?= instan (p X) a (all x\ p x).
Not LLambda unification encountered: a
```

Future developments: runtime errors

```
Kind tm, fm   type.
Type all   (tm -> fm) -> fm.
Type p         tm -> fm.
Type a         tm.
Define instan : fm -> tm -> fm -> prop by
   instan X T Y := (X = B T) /\ (Y = all B).

?= instan (p X) a (all x\ p x).
Not LLambda unification encountered: a
```

Future developments: runtime errors

```
Kind tm, fm   type.
Type all   (tm -> fm) -> fm.
Type p         tm -> fm.
Type a         tm.
Define instan : fm -> tm -> fm -> prop by
   instan X T Y := (Y = all B) /\ (X = B T).

?= instan (p X) a (all x\ p x).
Solution found:
 X = a
More [y] ?
No more solutions.
```

Future developments

- minimal vs nominal nabla
- Abella
- tables
- runtime errors (higher order pattern unification)

Future developments

- minimal vs nominal nabla
- Abella
- tables
- runtime errors (higher order pattern unification)
- front-ends

It's still a release *candidate*!

D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu.
The bedwyr system for model checking over syntactic expressions.
In *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction*, CADE-21, pages 391–397, Berlin, Heidelberg, 2007. Springer-Verlag.

D. Miller and A. Tiu.
A proof theory for generic judgments.
ACM Trans. on Computational Logic, 6(4):749–783, Oct. 2005.

A. Tiu.
A Logical Framework for Reasoning about Logical Specifications.
PhD thesis, Pennsylvania State University, May 2004.