Nijn/ONijn: A New Certification Engine for Higher-Order Termination

Cynthia Kop, **Deivid Vale**, and Deivid Vale

International Workshop on Higher-Order Rewriting
Rome, Italy
July 04th, 2023

Summary

What do I mean by Higher-Order Rewriting?

Roughly, a style of simply-typed λ-calculae extended with a set of type-annotated symbols.

What do I mean by Higher-Order Rewriting?

Roughly, a style of simply-typed λ-calculae extended with a set of type-annotated symbols.

$$\mathcal{R} := \begin{cases} \mathsf{map}\ F\ \mathsf{nil} \to \mathsf{nil} \\ \mathsf{map}\ F\ x :: xs \to (F\ x) :: \mathsf{map}\ F\ xs \end{cases}$$

What do I mean by Higher-Order Rewriting?

Roughly, a style of simply-typed λ-calculae extended with a set of type-annotated symbols.

$$\mathcal{R} := \begin{cases} \mathsf{map}\, F\, \mathsf{nil} \to \mathsf{nil} \\ \mathsf{map}\, F\, x :: xs \to (F\, x) :: \mathsf{map}\, F\, xs \end{cases}$$

$$\mathsf{map}\,(\lambda y.\mathsf{f}\, s\, x)\,[\ell_1; \ldots, \ell_k]$$

Termination Tools

- Proving termination is usually difficult to do in practice

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs
  - ...

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs
  - ...
- most termination techniques are built with automation in mind

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs
  - ...
- most termination techniques are built with automation in mind
- tools were built over the years to automate this search for termination proofs

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs
  - ...
- most termination techniques are built with automation in mind
- tools were built over the years to automate this search for termination proofs
  - AProVE, $\mathrm{T_TT_2}$, NaTT, SOL, Wanda, ...

Termination Tools

- Proving termination is usually difficult to do in practice
  - interpretation based techniques
  - rule removal
  - HORPO
  - dependency pairs
  - ...
- most termination techniques are built with automation in mind
- tools were built over the years to automate this search for termination proofs
  - AProVE, $\mathsf{T_TT_2}$, NaTT, SOL, Wanda, ...
- they compete annually in the termination competition

Problem Setting

- bugs in termination provers are usually **very** difficult to find

Problem Setting

- bugs in termination provers are usually **very** difficult to find
- and it occurs often

Problem Setting

- bugs in termination provers are usually **very** difficult to find
- and it occurs often

Problem Setting

- bugs in termination provers are usually **very** difficult to find
- and it occurs often

Our goal

provide guarantees that the outputted (informal proof) of termination tools (for higher-order rewriting) are correct

What did we do?

We introduce Nijn/ONijn. It includes:

- the formalization engine

What did we do?

We introduce Nijn/ONijn. It includes:

- the formalization engine
  - a formalization in Coq of the theory of higher-order rewriting

What did we do?

We introduce Nijn/ONijn. It includes:

- the formalization engine
  - a formalization in Coq of the theory of higher-order rewriting
  - a formalization of higher-order polynomial interpretation

What did we do?

We introduce Nijn/ONijn. It includes:

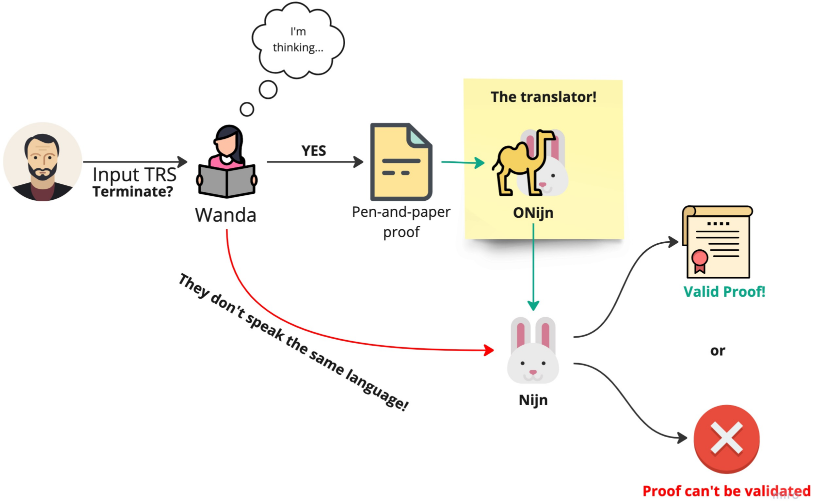
- the formalization engine
  - a formalization in Coq of the theory of higher-order rewriting
  - a formalization of higher-order polynomial interpretation
- the translation engine

What did we do?

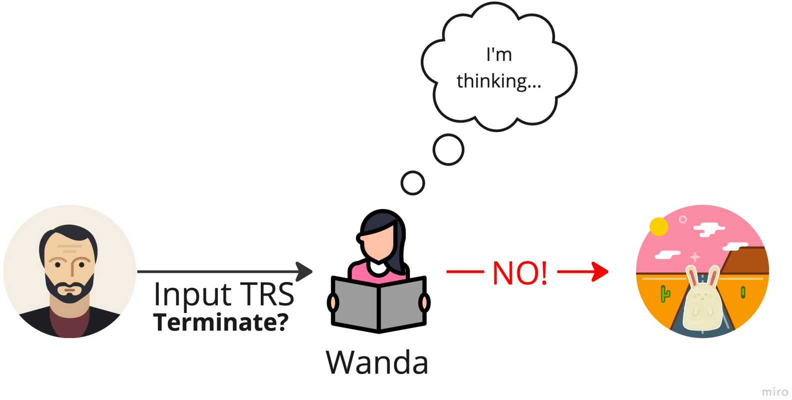
We introduce Nijn/ONijn. It includes:

- the formalization engine
  - a formalization in Coq of the theory of higher-order rewriting
  - a formalization of higher-order polynomial interpretation
- the translation engine
  - an OCaml program that turns the output of termination checkers (like Wanda) into a Coq script.

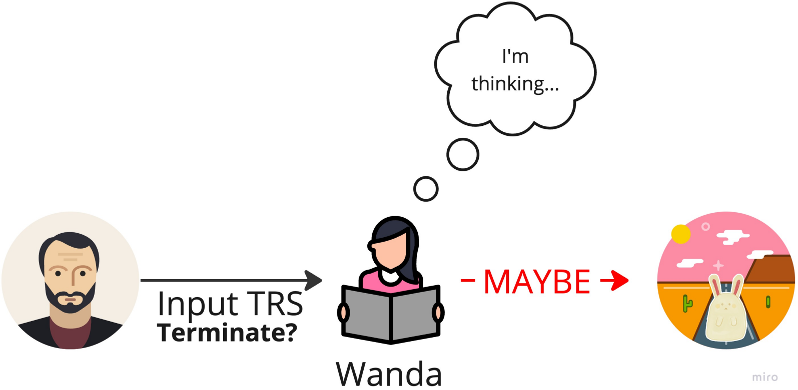
Mandatory picture, otherwise the talk is boring. . .



Mandatory picture, otherwise the talk is boring. . .



Mandatory picture, otherwise the talk is boring. . .



Nijn Formalization Statistics

```
spec     proof comments
5497      1985      272 total
```

Generation of proof-scripts — input TRS

```
nil : list
cons : (a * list) ⟶ list
map : (list * a → a) ⟶ list

x : a
l : list
F : a → a

map(nil, F) ⇒ nil
map(cons(x, l), F) ⇒ cons(F * x, map(l, F))

```

Generation of proof-scripts — Wanda's Output

Mixed_HO_10_map.onijn ×

experiments > ho_poly > Mixed_HO_10_map.onijn

You, 4 months ago | 1 author (You)

```
YES
Signature: [
  cons : a → list → list ;
  map : list → (a → a) → list ;
  nil : list
]

Rules: [
  map nil F ⇒ nil ;
  map (cons X Y) G ⇒ cons (G X) (map Y G)
]
```

Generation of proof-scripts — Wanda's Output

```
Interpretation: [
  J(cons) = Lam[y0;y1].3 + 2*y1 ;
  J(map) = Lam[y0;G1].3*y0 + 3*y0*G1(y0) ;
  J(nil) = 3
]

Removed: [
  map nil F ⇒ nil ;
  map (cons X Y) G ⇒ cons (G X) (map Y G)
]
```

ONijn's Output

coq_certificates > Mixed_HO_10_map.v

```
Require Import Nijn.Nijn.
Open Scope poly_scope.

Inductive base_types :=
| Ca
| Clist.
Global Instance decEq_base_types : decEq base_types.
Proof.
decEq_finite.
Defined.

Definition a :=
Base Ca.
Definition list :=
Base Clist.
```

ONijn's Output

```
Inductive fun_symbols :=
| Tcons
| Tmap
| Tnil.
Global Instance decEq_fun_symbols : decEq fun_symbols.
Proof.
decEq_finite.
Defined.
```

ONijn's Output

```
Definition fn_arity fn_symbols :=
match fn_symbols with
  | Tcons  ⇒  a → list → list
  | Tmap  ⇒  list → (a → a) → list
  | Tnil ⇒ list
end.
Definition cons {C} : tm fn_arity C _ :=
BaseTm Tcons.
Definition map {C} : tm fn_arity C _ :=
BaseTm Tmap.
Definition nil {C} : tm fn_arity C _ :=
BaseTm Tnil.
```

ONijn's Output

```
Program Definition rule_0 :=
    make_rewrite
    (_ ,, ·) _
    (map · nil · V 0)
    nil.
Program Definition rule_1 :=
    make_rewrite
    (_ ,, _ ,, _ ,, ·) _
    (map · (cons · V 0 · V 1) · V 2)
    (cons · ( V 2 · V 0) · (map · V 1 · V 2)).

Definition trs :=
  make_afs
    fn_arity
    (rule_0 :: rule_1 :: List.nil).

```

ONijn's Output

```

Definition map_fun_poly fn_symbols : poly · (arity trs fn_symbols) :=
match fn_symbols with
| Tcons ⇒
 λP
λP let y1 := P_var Vz in
(to_Poly (P_const 3 + P_const 2 * y1))
| Tmap ⇒
 λP let y0 := P_var (Vs Vz) in
λP let G1 := P_var Vz in
(to_Poly (P_const 3 * y0 + P_const 3 * y0 * (G1 ·P (y0))))
| Tnil ⇒
(to_Poly (P_const 3))
end.
Definition trs_isSN : isSN trs.
Proof.
solve_poly_SN map_fun_poly.
Qed.
```

ONijn Translator Statistics

Languages

language	files	code	comment	blank	total
OCaml	15	880	154	220	1,254
OCaml Interface	10	173	463	154	790
OCamldoc	1	108	0	32	140
OCamllex	1	16	32	3	51
Menhir	1	107	10	30	147
Shell Script	1	57	5	10	72

Conclusion

- We made a formalization with the basic results of higher order rewriting.

Conclusion

- We made a formalization with the basic results of higher order rewriting.
- We also formalized the polynomial method.

Conclusion

- We made a formalization with the basic results of higher order rewriting.
- We also formalized the polynomial method.
- We made an OCaml program that turned the output of a termination checker into a Coq script.

Conclusion

- We made a formalization with the basic results of higher order rewriting.
- We also formalized the polynomial method.
- We made an OCaml program that turned the output of a termination checker into a Coq script.
- The certification method is effective: we could verify the output of Wanda on a set of 46 problems.