Polyregular Model Checking

Aliaume Lopez and Rafał Stefański
University of Warsaw (now. LaBRI !)

Bordeaux
M2F Seminar, 2025-09-30

https://www.irif.fr/~alopez/

(Regular) Model Checking

Verify that a system satisfies a specification

(Regular) Model Checking

Verify that a system satisfies a specification

precondition $\{\varphi\}$ P $\{\psi\}$ postcondition

program

Hoare Triple

(Regular) Model Checking

Verify that a system satisfies a specification

precondition $\{\varphi\}$ P $\{\psi\}$ postcondition

program

Hoare Triple

Regular Model Checking :

- **Programs** : string-to-string programs
- **Specifications** : regular expressions

(Regular) Model Checking

Verify that a system satisfies a specification

precondition $\{\varphi\}$ P $\{\psi\}$ postcondition

program

HOARE TRIPLE

Regular Model Checking :

— **Programs** : string-to-string programs
— **Specifications** : regular expressions

Continuous functions :
$f^{-1}(L)$ is regular whenever L is a regular language

(Regular) Model Checking

Verify that a system satisfies a specification

precondition $\{\varphi\}$ P $\{\psi\}$ postcondition

Regular Model Checking of Continuous Functions

$$\varphi \implies P^{-1}(\psi)$$

Regular Model Checking :

- **Programs** : string-to-string programs
- **Specifications** : regular expressions

Continuous functions :
$f^{-1}(L)$ is regular whenever L is a regular language

(Regular) Model Checking

Verify that a system satisfies a specification

precondition $\{\varphi\}$ P $\{\psi\}$ postcondition

Regular Model Checking of Continuous Functions

$$\varphi \implies P^{-1}(\psi)$$

Regular Model Checking :

- **Programs** : string-to-string programs
- **Specifications** : regular expressions

Continuous functions :
$f^{-1}(L)$ is regular whenever L is a regular language

Continuity Quizz

Continuity Quizz

$$w \mapsto w\#w$$

Continuity Quizz

$$w \mapsto w\#w$$

Continuity Quizz

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$w \mapsto w\#w$

Continuity Quizz

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w \mapsto w\#w$$

Continuity Quizz

$$w \mapsto w^{|w|}$$

$w \mapsto w\#w$

$w_1\#w_2 \mapsto (w_1 = w_2)$

Continuity Quizz

$$w \mapsto w^{|w|}$$

$$w \mapsto w\#w \qquad w_1\#w_2 \mapsto (w_1 = w_2)$$

Continuity Quizz

$$w \mapsto \mathsf{sort}(w)$$

$w \mapsto w\#w$

$w_1\#w_2 \mapsto (w_1 = w_2)$

$w \mapsto w^{|w|}$

Continuity Quizz

$$w \mapsto \mathsf{sort}(w)$$

$$w \mapsto w\#w$$

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w \mapsto w^{|w|}$$

Continuity Quizz

$$w_1\#\cdots\#w_n \mapsto \mathsf{sort}(\cdots)$$

$$w \mapsto w\#w$$

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w \mapsto w^{|w|}$$

$$w \mapsto \mathsf{sort}(w)$$

Continuity Quizz

$$w_1\# \cdots \# w_n \mapsto \mathsf{sort}(\cdots)$$

$$w \mapsto w\#w$$

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w \mapsto w^{|w|}$$

$$w \mapsto \mathsf{sort}(w)$$

Continuity Quizz

$$w \mapsto a^{!|w|}$$

$$w \mapsto w\#w$$

$$w \mapsto w^{|w|}$$

$$w \mapsto \mathsf{sort}(w)$$

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w_1\# \cdots \#w_n \mapsto \mathsf{sort}(\cdots)$$

Continuity Quizz

$$w \mapsto a^{!|w|}$$

$$w \mapsto w\#w$$

$$w_1\#w_2 \mapsto (w_1 = w_2)$$

$$w \mapsto w^{|w|}$$

$$w_1\#\cdots\#w_n \mapsto \mathsf{sort}(\cdots)$$

$$w \mapsto \mathsf{sort}(w)$$

Continuity Quizz

$\exists f$ non-computable and continuous

$w \mapsto w\#w$

$w \mapsto w^{|w|}$

$w \mapsto \mathsf{sort}(w)$

$w \mapsto a^{!|w|}$

$w_1\#w_2 \mapsto (w_1 = w_2)$

$w_1\# \cdots \#w_n \mapsto \mathsf{sort}(\cdots)$

Continuity Quizz

$\exists f$ non-computable and continuous

$w \mapsto w\#w$

$w \mapsto w^{|w|}$

$w \mapsto \mathsf{sort}(w)$

$w \mapsto a^{!|w|}$

$w_1\#w_2 \mapsto (w_1 = w_2)$

$w_1\# \cdots \#w_n \mapsto \mathsf{sort}(\cdots)$

A zoo of transducer models

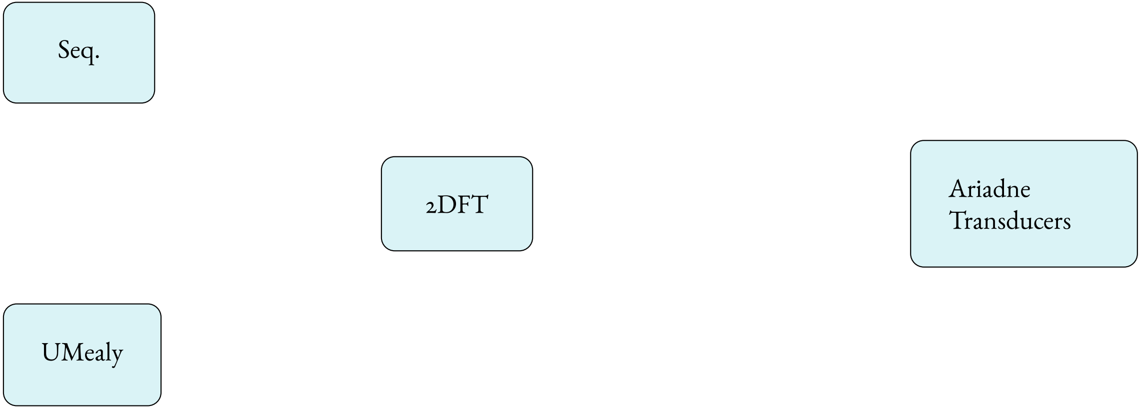
A zoo of transducer models

Ariadne
Transducers

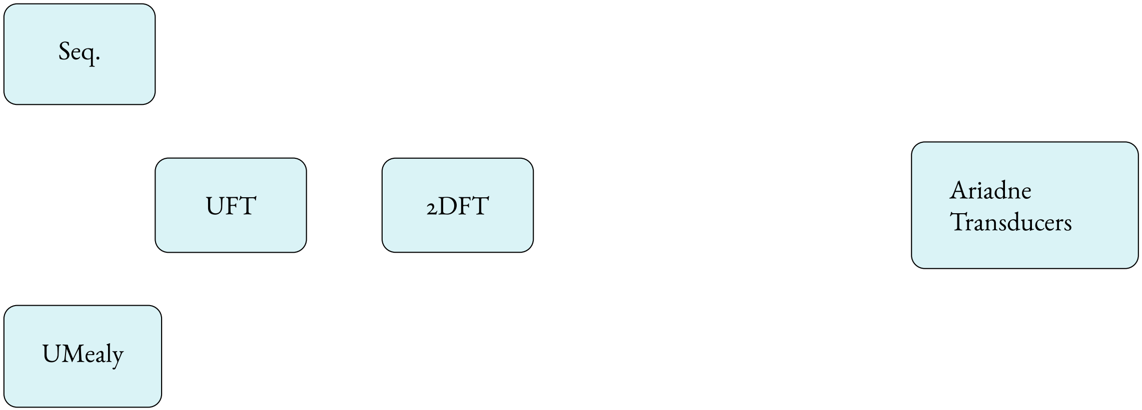
A zoo of transducer models

A zoo of transducer models

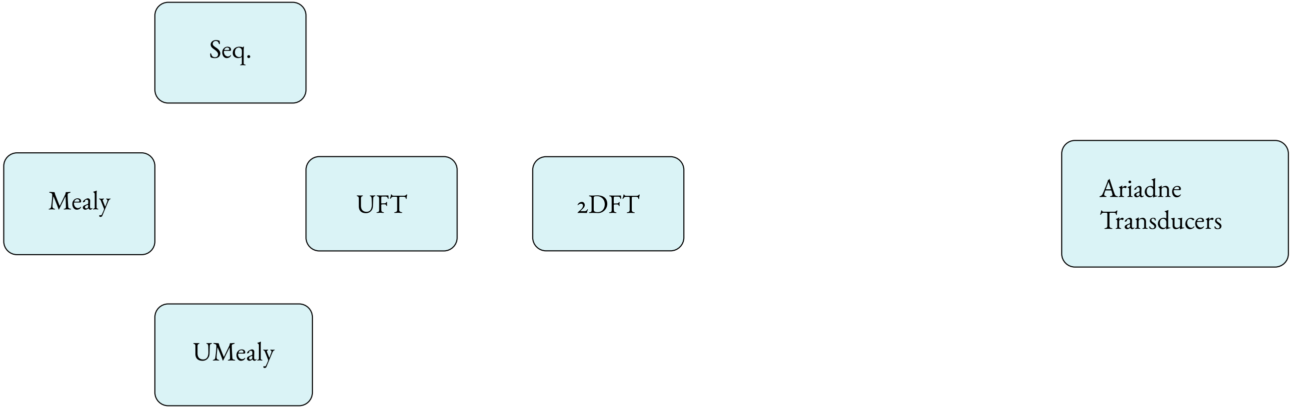
A zoo of transducer models



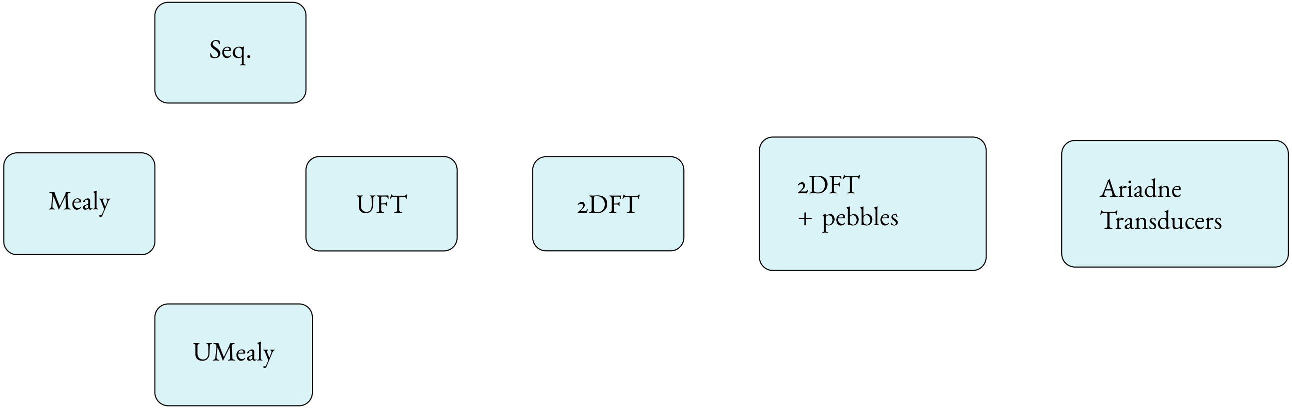
A zoo of transducer models



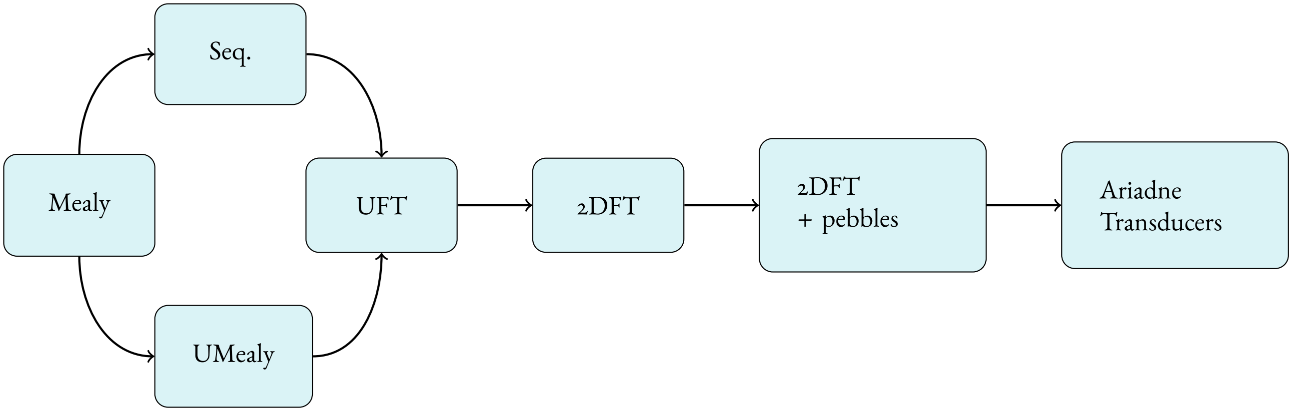
A zoo of transducer models



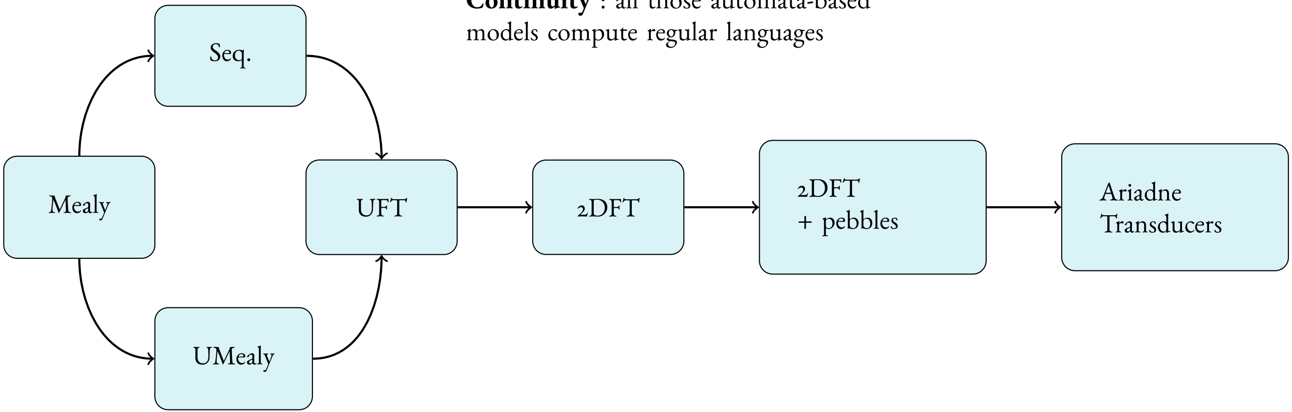
A zoo of transducer models



A zoo of transducer models

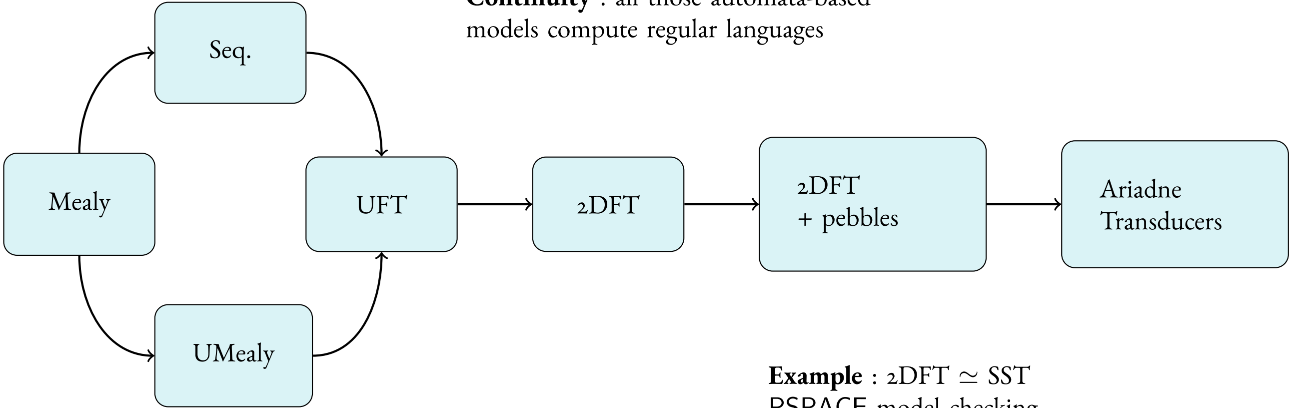


A zoo of transducer models



Continuity : all those automata-based models compute regular languages

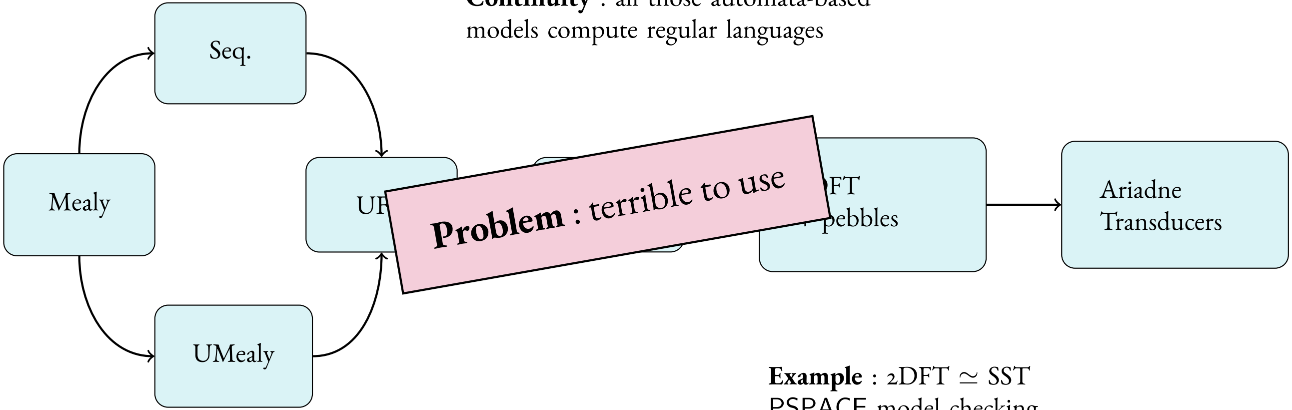
A zoo of transducer models



Continuity : all those automata-based models compute regular languages

Example : 2DFT $\simeq$ SST
PSPACE model checking
[Alur Černý, POPL'11]

A zoo of transducer models



Continuity : all those automata-based models compute regular languages

Example : 2DFT $\simeq$ SST
PSPACE model checking
[Alur Černý, POPL'11]

Polyregular Functions

MSO interpretations

For-programs

Pebble transducers

List functions

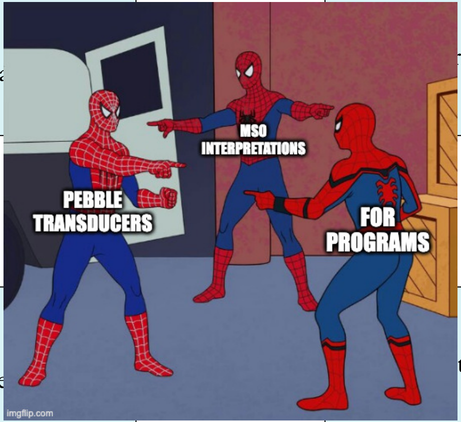
Polyregular Functions

MSO
interpreta

rams

Pebble
transduce

tions



Let's design

```python
def getBetween(l, i, j):
    """ Get elements between i and j """
    for (k, c) in enumerate(l):
        if i <= k and k <= j: ①
            yield c ②

def containsAB(w):
    """ Contains "ab" as a subsequence """
    seen_a = False ③
    for (x, c) in enumerate(w):
        if c == "a": ④
                seen_a = True ⑤
        elif seen_a and c == "b":
            return True
    return False

def subwordsWithAB(word):
    """ Get subwords that contain "ab" """
    for (i,c) in enumerate(word): ⑥
        for (j,d) in reversed(enumerate(word)): ⑦
            s = getBetween(word, i, j) ⑧
            if containsAB(s):
                yield s
```

Fig. 1. A small Python program that outputs all subwords of a given word containing `ab` as a scattered subword

Let's design

Rules of the fight

- **lists** (2)
- **loops** (6) and (7)
- **variables** (6)
- **equality** (4)
- **tests** (1)
- **shadowing** no nay never
- **functions** no boolean inputs
- **updates** (3) and (5)

```
def getBetween(l, i, j):
    """ Get elements between i and j """
    for (k, c) in enumerate(l):
        if i <= k and k <= j: ①
            yield c ②

def containsAB(w):
    """ Contains "ab" as a subsequence """
    seen_a = False ③
    for (x, c) in enumerate(w):
        if c == "a": ④
                seen_a = True ⑤
        elif seen_a and c == "b":
            return True
    return False

def subwordsWithAB(word):
    """ Get subwords that contain "ab" """
    for (i,c) in enumerate(word): ⑥
        for (j,d) in reversed(enumerate(word)): ⑦
            s = getBetween(word, i, j) ⑧
            if containsAB(s):
                yield s
```

Fig. 1. A small Python program that outputs all subwords of a given word containing ab as a scattered subword

Let's design

Rules of the fight

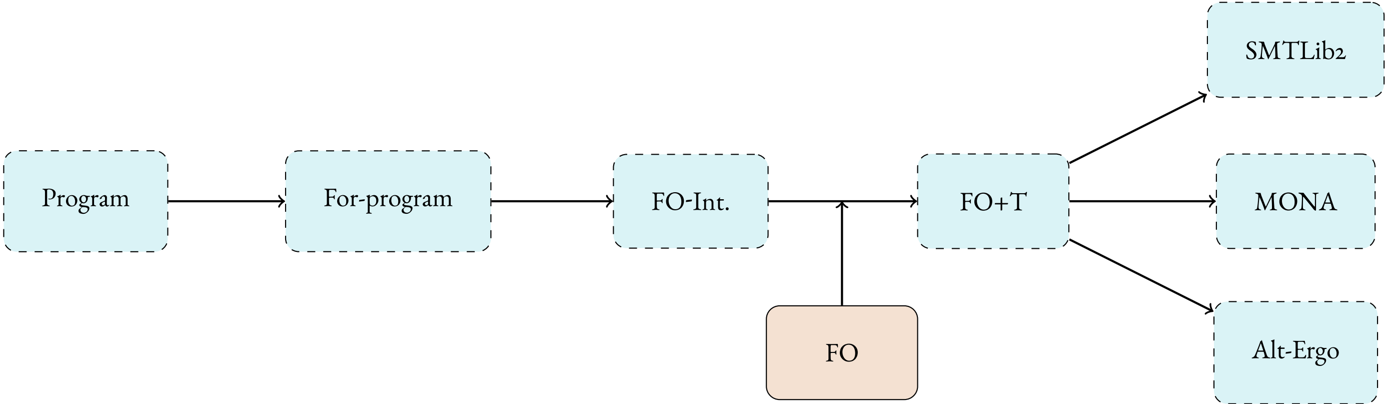
- **lists** (2)
- **loops** (6) and (7)
- **variables** (6)
- **equality** (4)
- **tests** (1)
- **shadowing** no nay never
- **functions** no boolean inputs
- **updates** (3) and (5)

```
def getBetween(l, i, j):
    """ Get elements between i and j """
    for (k, c) in enumerate(l):
        if i <= k and k <= j: (1)
            yield c (2)

def containsAB(w):
    """ Contains "ab" as a subsequence """
    seen_a = False (3)
                  erate(w):
                  )
                  = True (5)
                  d c == "b":
                  e



                  d):
                  hat contain "ab" """
    for (i,c) in enumerate(word): (6)
        for (j,d) in reversed(enumerate(word)): (7)
            s = getBetween(word, i, j) (8)
            if containsAB(s):
                yield s
```

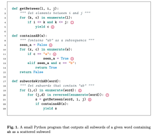
Fig. 1. A small Python program that outputs all subwords of a given word containing ab as a scattered subword

DEMO

Anatomy of a For(program checker)

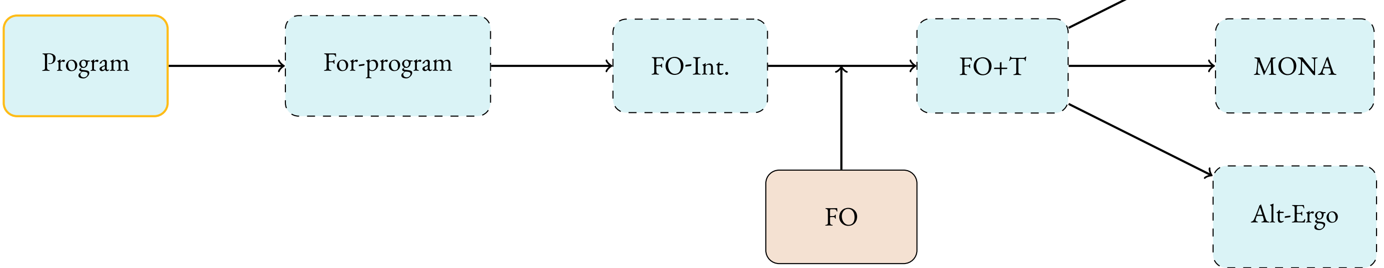


Anatomy of a For(program checker)



```
def getBetween(l, i, j):
    """ Get elements between i and j """
    for (k, c) in enumerate(l):
        if i <= k and k <= j: ①
            yield c ②

def containsAB(w):
    """ Contains "ab" as a subsequence """
    seen_a = False ③
    for (x, c) in enumerate(w):
        if c == "a": ④
                seen_a = True ⑤
        elif seen_a and c == "b":
            return True
    return False

def subwordsWithAB(word):
    """ Get subwords that contain "ab" """
    for (i,c) in enumerate(word): ⑥
        for (j,d) in reversed(enumerate(word)): ⑦
            s = getBetween(word, i, j) ⑧
            if containsAB(s):
                yield s
```

Fig. 1. A small Python program that outputs all subwords of a given word containing ab as a scattered subword



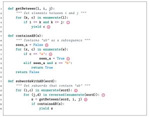
Simple for programs

Rules of the fight

- **functions** no no no
- **lists** no!
- **variables** only booleans / positions

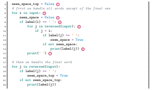
```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥
        print(' ') ⑦

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))
```

Anatomy of a For(program checker)

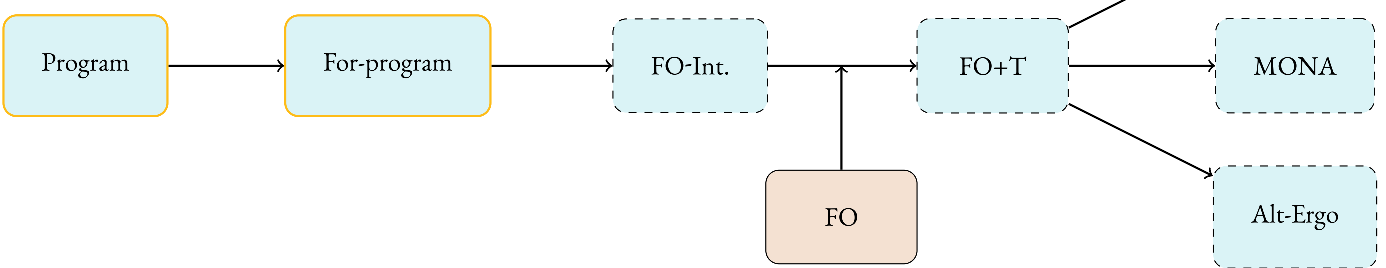


```python
def getBetween(l, i, j):
    """ Get elements between i and j """
    for (k, c) in enumerate(l):
        if i <= k and k <= j:
            yield c

def containsAB(w):
    """ Contains "ab" as a subsequence """
    seen_a = False
    for (x, c) in enumerate(w):
        if c == "a":
                seen_a = True
        elif seen_a and c == "b":
            return True
    return False

def subwordsWithAB(word):
    """ Get subwords that contain "ab" """
    for (i,c) in enumerate(word):
        for (j,d) in reversed(enumerate(word)):
            s = getBetween(word, i, j)
            if containsAB(s):
                yield s
```

Fig. 1. A small Python program that outputs all subwords of a given word containing ab as a scattered subword



```python
seen_space_top = False
# first we handle all words except of the final one
for i in input:
    seen_space = False
    if label(i) == ' ':
        for j in reversed(input):
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j))
        print(' ')

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))
```



First-Order Interpretations

First-Order Interpretations

Rules of the fight

tags finite set tags

arities $\mathsf{ar} \colon \mathsf{tags} \to \mathbb{N}$

domain first order formulas φ^t_{dom}

output letters $\mathsf{out} \colon \mathsf{tags} \to A + \mathbb{N}$

output order first order formulas $\varphi^{t_1,t_2}_{\leq}$

First-Order Interpretations

Rules of the fight

tags finite set tags

arities $\mathsf{ar}\colon \mathsf{tags} \to \mathbb{N}$

domain first order formulas φ^t_{dom}

output letters $\mathsf{out}\colon \mathsf{tags} \to A + \mathbb{N}$

output order first order formulas $\varphi^{t_1,t_2}_{\leq}$

$\mathrm{out}(\texttt{printB}) = \texttt{b} \quad \mathrm{out}(\texttt{copy}) = 1$

$\varphi^{\texttt{printB}}_{\mathrm{dom}}(x) : x =_L \texttt{b} \quad \varphi^{\texttt{copy}}_{\mathrm{dom}}(x) : x \neq_L \texttt{b}$

$\varphi_{\leq}$	$\texttt{printB}(x_1)$	$\texttt{copy}(x_1)$
$\texttt{printB}(x_2)$	$x_1 \leq x_2$	$x_1 < x_2$
$\texttt{copy}(x_2)$	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The `swapAsToBs` interpretation.

First-Order Interpretations

Rules of the fight

- **tags** finite set tags
- **arities** $\mathsf{ar}\colon \mathsf{tags} \to \mathbb{N}$
- **domain** first order formulas φ^t_{dom}
- **output letters** $\mathsf{out}\colon \mathsf{tags} \to A + \mathbb{N}$
- **output order** first order formulas $\varphi^{t_1,t_2}_{\leq}$

a u t o m a t e s
0 1 2 3 4 5 6 7 8

$$\mathrm{out}(\mathtt{printB}) = \mathtt{b} \qquad \mathrm{out}(\mathtt{copy}) = 1$$

$$\varphi^{\mathtt{printB}}_{\mathrm{dom}}(x) : x =_L \mathtt{b} \qquad \varphi^{\mathtt{copy}}_{\mathrm{dom}}(x) : x \neq_L \mathtt{b}$$

$\varphi_\leq$	$\mathtt{printB}(x_1)$	$\mathtt{copy}(x_1)$
$\mathtt{printB}(x_2)$	$x_1 \leq x_2$	$x_1 < x_2$
$\mathtt{copy}(x_2)$	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The `swapAsToBs` interpretation.

First-Order Interpretations

Rules of the fight

tags finite set tags

arities ar: tags $\to \mathbb{N}$

domain first order formulas $\varphi^{t}_{\mathsf{dom}}$

output letters out: tags $\to A + \mathbb{N}$

output order first order formulas $\varphi^{t_1,t_2}_{\leq}$

	a	u	t	o	m	a	t	e	s
	0	1	2	3	4	5	6	7	8
printB	0	1	2	3	4	5	6	7	8
copy	0	1	2	3	4	5	6	7	8

$\mathrm{out}(\texttt{printB}) = \texttt{b} \quad \mathrm{out}(\texttt{copy}) = 1$

$\varphi^{\texttt{printB}}_{\mathrm{dom}}(x) : x =_L \texttt{b} \quad \varphi^{\texttt{copy}}_{\mathrm{dom}}(x) : x \neq_L \texttt{b}$

$\varphi_{\leq}$	printB(x_1)	copy(x_1)
printB(x_2)	$x_1 \leq x_2$	$x_1 < x_2$
copy(x_2)	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The swapAsToBs interpretation.

First-Order Interpretations

Rules of the fight

- **tags** finite set tags
- **arities** $\mathsf{ar}\colon \mathsf{tags} \to \mathbb{N}$
- **domain** first order formulas $\varphi^{t}_{\mathsf{dom}}$
- **output letters** $\mathsf{out}\colon \mathsf{tags} \to A + \mathbb{N}$
- **output order** first order formulas $\varphi^{t_1,t_2}_{\leq}$

	a	u	t	o	m	a	t	e	s
	0	1	2	3	4	5	6	7	8
printB	0	1	2	3	4	5	6	7	8
copy	0	1	2	3	4	5	6	7	8

$\mathrm{out}(\mathtt{printB}) = \mathtt{b}$ $\quad$ $\mathrm{out}(\mathtt{copy}) = 1$

$\varphi^{\mathtt{printB}}_{\mathrm{dom}}(x) : x =_L \mathtt{b}$ $\quad$ $\varphi^{\mathtt{copy}}_{\mathrm{dom}}(x) : x \neq_L \mathtt{b}$

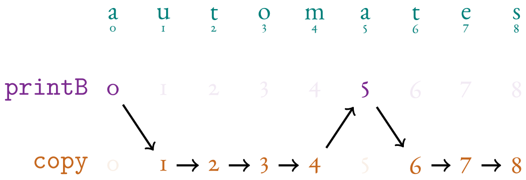
$\varphi_{\leq}$	$\mathtt{printB}(x_1)$	$\mathtt{copy}(x_1)$
$\mathtt{printB}(x_2)$	$x_1 \leq x_2$	$x_1 < x_2$
$\mathtt{copy}(x_2)$	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The `swapAsToBs` interpretation.

First-Order Interpretations

Rules of the fight

- **tags** finite set tags
- **arities** $\mathsf{ar} \colon \mathsf{tags} \to \mathbb{N}$
- **domain** first order formulas φ^t_{dom}
- **output letters** $\mathsf{out} \colon \mathsf{tags} \to A + \mathbb{N}$
- **output order** first order formulas $\varphi^{t_1,t_2}_{\leq}$



$\mathsf{out}(\mathtt{printB}) = \mathtt{b}$ $\quad \mathsf{out}(\mathtt{copy}) = 1$

$\varphi^{\mathtt{printB}}_{\mathrm{dom}}(x) : x =_L \mathtt{b}$ $\quad \varphi^{\mathtt{copy}}_{\mathrm{dom}}(x) : x \neq_L \mathtt{b}$

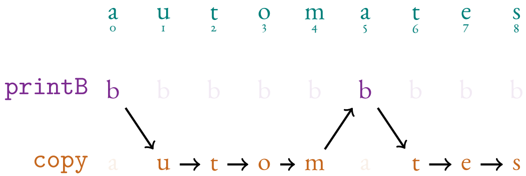
$\varphi_{\leq}$	$\mathtt{printB}(x_1)$	$\mathtt{copy}(x_1)$
$\mathtt{printB}(x_2)$	$x_1 \leq x_2$	$x_1 < x_2$
$\mathtt{copy}(x_2)$	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The `swapAsToBs` interpretation.

First-Order Interpretations

Rules of the fight

- **tags** finite set tags
- **arities** $\mathsf{ar}\colon \mathsf{tags} \to \mathbb{N}$
- **domain** first order formulas φ^t_{dom}
- **output letters** $\mathsf{out}\colon \mathsf{tags} \to A + \mathbb{N}$
- **output order** first order formulas $\varphi^{t_1,t_2}_{\leq}$



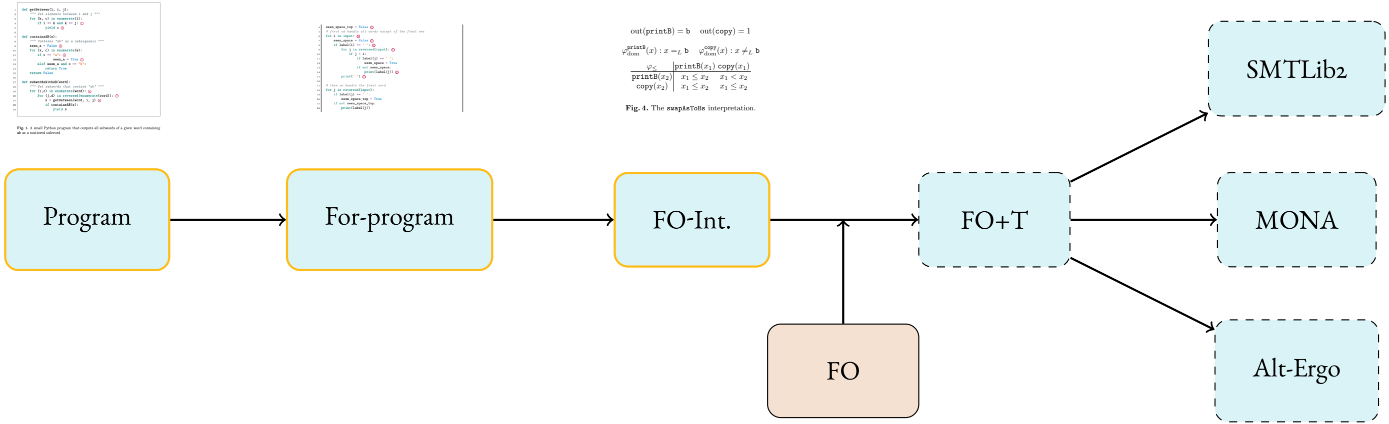
$\mathrm{out}(\texttt{printB}) = \mathtt{b} \quad \mathrm{out}(\texttt{copy}) = 1$

$\varphi^{\texttt{printB}}_{\mathrm{dom}}(x) : x =_L \mathtt{b} \quad \varphi^{\texttt{copy}}_{\mathrm{dom}}(x) : x \neq_L \mathtt{b}$

$\varphi_{\leq}$	$\texttt{printB}(x_1)$	$\texttt{copy}(x_1)$
$\texttt{printB}(x_2)$	$x_1 \leq x_2$	$x_1 < x_2$
$\texttt{copy}(x_2)$	$x_1 \leq x_2$	$x_1 \leq x_2$

Fig. 4. The `swapAsToBs` interpretation.

Anatomy of a For(program checker)



First-Order Logic… with tags!

$\varphi := \exists x : \mathsf{tag}, \varphi \mid \exists x : \mathsf{pos}, \varphi \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid x =_L \mathsf{a} \mid x \leq y \mid x = \mathsf{t}$

First-Order Logic... with tags!

$\varphi := \exists x : \mathsf{tag}, \varphi \mid \exists x : \mathsf{pos}, \varphi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid x =_L \mathsf{a} \mid x \leq y \mid x = \mathsf{t}$

$$\mathsf{FO} \xrightarrow{f \in \mathsf{FO\text{-}I}} \mathsf{FO{+}T}$$

$$\forall w, f(w) \models \varphi \iff w \models f(\varphi)$$

First-Order Logic… with tags!

$$\varphi := \exists x : \mathsf{tag}, \varphi \mid \exists x : \mathsf{pos}, \varphi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid x =_L \mathsf{a} \mid x \leq y \mid x = \mathsf{t}$$

$f \in$ FO-I

$$\mathsf{FO} \xrightarrow{\quad f \in \mathsf{FO\text{-}I} \quad} \mathsf{FO{+}T}$$

$$\forall w, f(w) \models \varphi \iff w \models f(\varphi)$$

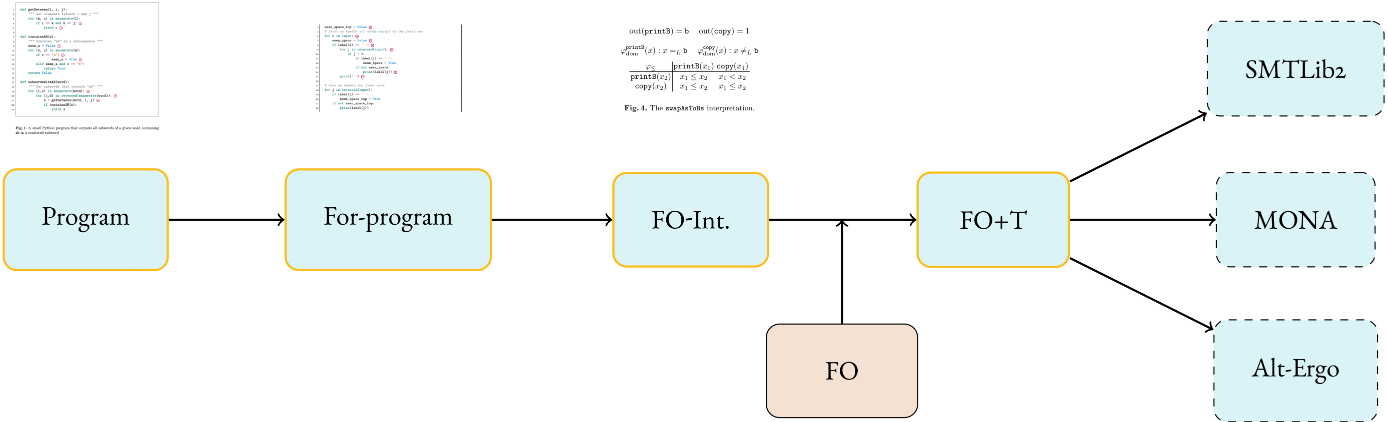
$$f(\forall_x \psi) := \forall_{t_x \in \mathsf{tags}} \forall_{x_1, \dots, x_{\mathsf{ar}(f)}} \left(\mathsf{dom}(t_x, x_1, \dots, x_{\mathsf{ar}(f)}) \Rightarrow f(\psi)\right)$$

$$\mathsf{dom}(t, x_1, \dots, x_{\mathsf{ar}(t)}) := \bigvee_{t' \in \mathsf{tags}} \left(t = t' \wedge \varphi^{t'}_{\mathsf{dom}}(x_1, \dots, x_{\mathsf{ar}(t')})\right)$$

$$f(x \leq y) := \bigvee_{t_1, t_2 \in \mathsf{tags}} \left(t_x = t_1 \wedge t_y = t_2 \wedge \varphi^{t_1, t_2}_{\leq}(x_1, \dots, x_{\mathsf{ar}(t_1)}, y_1, \dots, y_{\mathsf{ar}(t_2)})\right)$$

$$f(x =_L \mathsf{a}) := \left(\bigvee_{t \in \mathsf{tags} \wedge \mathsf{out}(t) = \mathsf{a}} t = t_x\right) \vee \left(\bigvee_{t \in \mathsf{tags} \wedge \mathsf{out}(t) \notin A} (t = t_x \wedge x_{\mathsf{out}(t)} =_L \mathsf{a})\right)$$

Anatomy of a For(program checker)

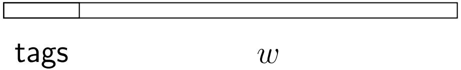


Calling solvers for help

Calling solvers for help

MONA

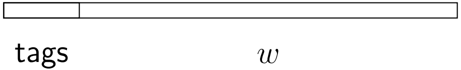
Solves : WS1S/WS2S over words



Calling solvers for help

MONA

Solves : WS1S/WS2S over words



SMTLib2

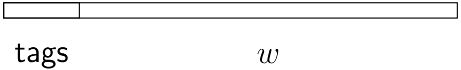
Solves : First order theories

- `DT` : `tags`
- `UF` : $w : \mathbb{N} \to A + \bot$
- `LIA` : positions

CALLING SOLVERS FOR HELP

MONA

Solves : WS1S/WS2S over words



Complete but slow

SMTLIB2

Solves : First order theories

- `DT` : `tags`
- `UF` : $w : \mathbb{N} \to A + \bot$
- `LIA` : positions

Incomplete but fast

Calling solvers for help

MONA

Solves : WS1S/WS2S over words

tags w

Complete but sl

SMTLib2

Solves : First order theories

— DT : tags
— UF : $w : \mathbb{N} \to A + \bot$
: positions

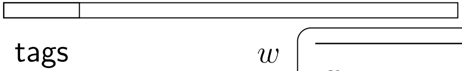
omplete but fast

filename	FP size	FP l.d.	FP b.d.	S.FP size	S.FP l.d.	S.FP b.d.	FO-I size	FO-I q.r.
identity.pr	3	1	0	2	2	0	1	0
reverse.pr	3	1	0	2	2	0	1	0
subwords_ab.pr	24	2	1	15	4	3	956	14
map_reverse.pr	36	2	1	18	4	1	285	5
prefixes.pr	6	2	0	5	3	0	2	0
get_last_word.pr	18	1	1	23	4	2	8553	15
get_first_word.pr	22	1	1	5	2	0	103	4
compress_as.pr	12	1	1	12	3	2	209	10
litteral_test.pr	29	1	1	129	3	12	3.2×10^4	82
bibtex.pr	110	2	1	802	6	29	13.7×10^6	136

Calling solvers for help

MONA

Solves : WS1S/WS2S over words



Complete but s

SMTLib2

Solves : First order theories

- DT : tags
- UF : $w \colon \mathbb{N} \to A + \bot$
- : positions

omplete but fast

filename	FP size	FP l.d.	FP b.d.	S.FP size	S.FP l.d.	S.FP b.d.	FO-I size	FO-I q.r.
identity.pr	3	1	0	2	2	0	1	0
reverse.pr	3	1	0	2	2	0	1	0
subwords_ab.pr	24	2	1	15	4	3	956	14
map_reverse.pr	36	2	1	18	4	1	285	5
prefixes.pr	6	2	0	5	3	0	2	0
get_last_word.pr	18	1	1	23	4	2	8553	15
get_first_word.pr	22	1	1	5	2	0	103	4
compress_as.pr	12	1	1	12	3	2	209	10
litteral_test.pr	29	1	1	129	3	12	3.2×10^4	82
bibtex.pr	110	2	1	802	6	29	13.7×10^6	136

FO model checking on words is TOWER-complete [Stockmeyer, 1974]

Anatomy of a For(program checker)

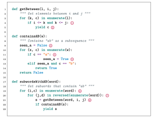
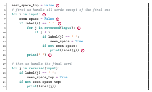


Fig. 1. A small Python program that outputs all subwords of a given word containing ab as a scattered subword



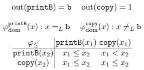
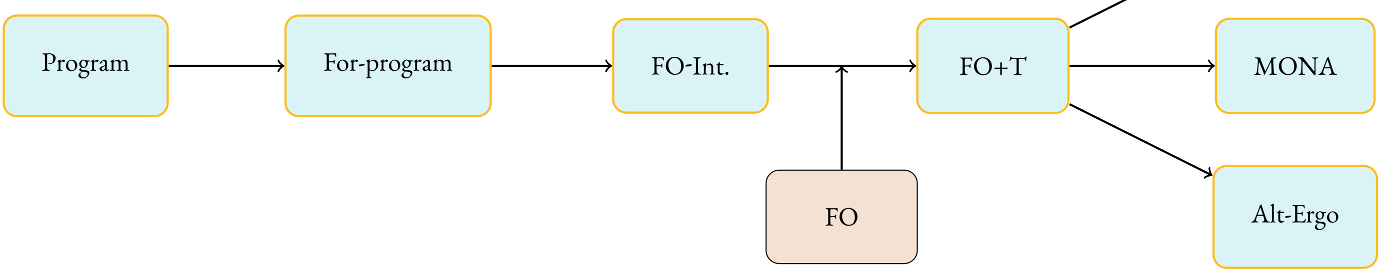


Fig. 4. The swapAsToBs interpretation.



Compiling to First Order

```python
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥
        print(' ') ⑦

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))
```

Compiling to First Order

```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥    t1
        print(' ') ⑦                     t2

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))                  t3
```

Tags : tags $= \{t_1, t_2, t_3\}$

Compiling to First Order

```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥      t1
        print(' ') ⑦                       t2

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))                    t3
```

Tags : tags $= \{t_1, t_2, t_3\}$

Arities : $\mathsf{ar}(t_1) = 2, \mathsf{ar}(t_2) = 1, \mathsf{ar}(t_3) = 1$

Compiling to First Order

```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥      t1
        print(' ') ⑦                       t2

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))                    t3
```

Tags : $\text{tags} = \{t_1, t_2, t_3\}$

Arities : $\text{ar}(t_1) = 2, \text{ar}(t_2) = 1, \text{ar}(t_3) = 1$

Out : $\text{out}(t_1) = j$, $\text{out}(t_2) = \text{space}$, $\text{out}(t_3) = j$

Compiling to First Order

```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥        t1
        print(' ') ⑦                          t2

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))                       t3
```

Tags : $\mathsf{tags} = \{t_1, t_2, t_3\}$

Arities : $\mathsf{ar}(t_1) = 2, \mathsf{ar}(t_2) = 1, \mathsf{ar}(t_3) = 1$

Out : $\mathsf{out}(t_1) = j$, $\mathsf{out}(t_2) = \mathsf{space}$, $\mathsf{out}(t_3) = j$

Order : Lexicographic based on positions (QF)

Compiling to First Order

```
seen_space_top = False (1)
# first we handle all words except of the final one
for i in input: (2)
    seen_space = False (3)
    if label(i) == ' ': (4)
        for j in reversed(input): (5)
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) (6)      t1
        print(' ') (7)                       t2

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))                      t3
```

Tags : tags $= \{t_1, t_2, t_3\}$

Arities : $\mathsf{ar}(t_1) = 2, \mathsf{ar}(t_2) = 1, \mathsf{ar}(t_3) = 1$

Out : $\mathsf{out}(t_1) = j$, $\mathsf{out}(t_2) = \mathsf{space}$, $\mathsf{out}(t_3) = j$

Order : Lexicographic based on positions (QF)

Domain : ... difficult part!

values of the boolean variables?

Compiling to First Order

```
seen_space_top = False ①
# first we handle all words except of the final one
for i in input: ②
    seen_space = False ③
    if label(i) == ' ': ④
        for j in reversed(input): ⑤
            if j < i:
                if label(j) == ' ':
                    seen_space = True
                if not seen_space:
                    print(label(j)) ⑥
        print(' ') ⑦

# then we handle the final word
for j in reversed(input):
    if label(j) == ' ':
        seen_space_top = True
    if not seen_space_top:
        print(label(j))
```

t_3

Tags : tags $= \{t_1, t_2, t_3\}$

…, ar$(t_3) = 1$

…space, out$(t_3) = j$

… positions (QF)

…riables ?

Program formulas

- FO + input (pos,bool) / output (bool)
- Can be composed easily
- Can implement if-then-else
- Can implement loops

One can write a program formula to compute the boolean variables at a given program position.

From High to Low

From High to Low

New operator : generator expressions.

- $\mathsf{gen}(s)$
- Can be used in place of a list / boolean
- Captures list variables but not boolean variables
- Simulate function calls

From High to Low

New operator : generator expressions.

- gen(s)
- Can be used in place of a list / boolean
- Captures list variables but not boolean variables
- Simulate function calls

Example code :

```
for (i,x) in enumerate(gen( expr )) ...
```

From High to Low

New operator : generator expressions.

- $\mathsf{gen}(s)$
- Can be used in place of a list / boolean
- Captures list variables but not boolean variables
- Simulate function calls

Example code :

```
for (i,x) in enumerate(gen( expr )) ...
```

Easy rewriting steps :

1. Remove literals
2. Remove functions
3. Remove boolean generators
4. Remove let expressions
5. Push boolean introductions upwards

From High to Low

New operator : generator expressions.

- gen(s)
- Can be used in place of a list / boolean
- Captures list variables but not boolean variables
- Simulate function calls

Example code :

```
for (i,x) in enumerate(gen( expr )) ...
```

Easy rewriting steps :

1. Remove literals
2. Remove functions
3. Remove boolean generators
4. Remove let expressions
5. Push boolean introductions upwards

Print all but first, program "s"

```
b = False
for (i,x) in enumerate(u):
   if b:
      yield x
   else:
      b = True
```

What are the following programs doing?

```
for (i,x) in reverse(enumerate(s)):
   yield x
```

```
for (i,x) in enumerate(s):
   yield x
```

Forward Loop Elimination

```
for (i,x) in enumerate(s):
    for (j,y) in enumerate(s):
        if i == j:
            yield x
```

Forward Loop Elimination

```
for (i,x) in enumerate(s):
    for (j,y) in enumerate(s):
        if i == j:
            yield x
```

Idea : substitute the body of the loop in s.
$s[\mathsf{yield} e \mapsto \ldots]$.

Forward Loop Elimination

```
for (i,x) in enumerate(s):
  for (j,y) in enumerate(s):
    if i == j:
      yield x
```

Idea : substitute the body of the loop in s.
$s[\mathsf{yield} e \mapsto \ldots]$.

Problem : We lost the index variable i!

Forward Loop Elimination

```
for (i,x) in enumerate(s):
  for (j,y) in enumerate(s):
    if i == j:
      yield x
```

Idea : substitute the body of the loop in s.
$s[\mathsf{yield} e \mapsto \ldots]$.

Problem : We lost the index variable i!

Solution :

- i can only be used in tests
- i can only be tested against positions of s
- we can replace $i = j$ by an **order formula**

Forward Loop Elimination

```
for (i,x) in enumerate(s):
   for (j,y) in enumerate(s):
      if i == j:
         yield x
```

Idea : substitute the body of the loop in s.
$s[\mathsf{yield} e \mapsto \ldots]$.

Problem : We lost the index variable i!

Solution :

- i can only be used in tests
- i can only be tested against positions of s
- we can replace $i = j$ by an **order formula**

```
b1 = False
for (i1,x1) in enumerate(s):
   if b1:

   else:
      b1 = True
```

Forward Loop Elimination

```
for (i,x) in enumerate(s):
    for (j,y) in enumerate(s):
        if i == j:
            yield x
```

Idea : substitute the body of the loop in s.
$s[\text{yield}e \mapsto \dots]$.

Problem : We lost the index variable i!

Solution :

- i can only be used in tests
- i can only be tested against positions of s
- we can replace $i = j$ by an **order formula**

```
b1 = False
for (i1,x1) in enumerate(s):
    if b1:
        b2 = False
        for (i2,x2) in enumerate(s):
            if b2:

            else:
                b2 = True
    else:
        b1 = True
```

Forward Loop Elimination

```
for (i,x) in enumerate(s):
    for (j,y) in enumerate(s):
        if i == j:
            yield x
```

Idea : substitute the body of the loop in s.
s[yield$e \mapsto \ldots$].

Problem : We lost the index variable i!

Solution :

- i can only be used in tests
- i can only be tested against positions of s
- we can replace $i = j$ by an **order formula**

```
b1 = False
for (i1,x1) in enumerate(s):
    if b1:
        b2 = False
        for (i2,x2) in enumerate(s):
            if b2:
                if i1 = i2:
                    yield x1
            else:
                b2 = True
    else:
        b1 = True
```

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
    yield x
```

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
   yield x
```

Problem : reverse a non reversible computation !

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
   yield x
```

Problem : reverse a non reversible computation!

Solution :

- Compute a *superset* of the reachable yields in the reversed order
- For every yield, check that it would be reachable
- If so, perform the rest of the computation

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
    yield x
```

Problem : reverse a non reversible computation!

Solution :

- Compute a *superset* of the reachable yields in the reversed order
- For every yield, check that it would be reachable
- If so, perform the rest of the computation

Remark : *this is the proof that* polyregular functions *are closed under composition.*

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
   yield x
```

Problem : reverse a non reversible computation!

Solution :

- Compute a *superset* of the reachable yields in the reversed order
- For every yield, check that it would be reachable
- If so, perform the rest of the computation

Remark : *this is the proof that* polyregular functions *are closed under composition.*

```
for (i1, x1) in reversed(enumerate(u)):
```

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
  yield x
```

Problem : reverse a non reversible computation !

Solution :

- Compute a *superset* of the reachable yields in the reversed order
- For every yield, check that it would be reachable
- If so, perform the rest of the computation

```
for (i1, x1) in reversed(enumerate(u)):
  for (i2, x2) in enumerate(u):
    b2 = False
    if b2:

    else:
      b2 = True
```

Remark : *this is the proof that* polyregular functions *are closed under composition.*

Backward Loop Elimination

```
for (i,x) in reverse(enumerate(s)):
    yield x
```

Problem : reverse a non reversible computation !

Solution :

- Compute a *superset* of the reachable yields in the reversed order
- For every yield, check that it would be reachable
- If so, perform the rest of the computation

```
for (i1, x1) in reversed(enumerate(u)):
  for (i2, x2) in enumerate(u):
    b2 = False
    if b2:
      if i1 = i2:
        yield x1
    else:
      b2 = True
```

Remark : *this is the proof that* polyregular functions *are closed under composition.*

In the end...

CAV Artifact Evaluation ★ Available

Polyregular Model Checking

Aliaume Lopez[1] [0000-0002-4205-327X]⋆ and Rafał Stefański[1] [0000-0002-8439-4056]⋆⋆

University of Warsaw

CAV Artifact Evaluation ★★★ Reusable

And more :

- Haskell implementation + webapp
- Nix / Docker / reproducible builds
- Symbolic alphabets
- Some optimisations

Future work :

- Comparison with other models
- Better interface with solvers
- Composable checks
- Monadic second order logic