A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests

Tobias Kappé[1,2] Todd Schmid[3,*] Alexandra Silva[4]

[1]Open Universiteit

[2]ILLC, University of Amsterdam

[3]Saint Mary's College of California

[4]Cornell University

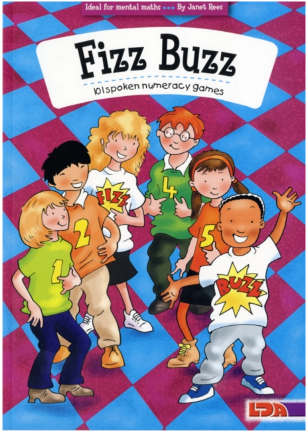
June 7, 2024

* who kindly let me use his slides

Reasoning about software: A story

Let's play a game of *Fizzbuzz!*

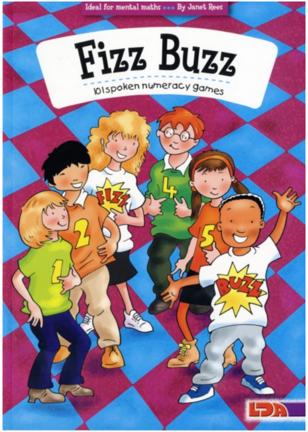
- Take turns counting to 100.
- Number divisible by 3 $\Rightarrow$ say *Fizz!*
- Number divisible by 5 $\Rightarrow$ say *Buzz!*
- Number is divisible by 3 *and* 5 $\Rightarrow$ say *Fizzbuzz!*
- Otherwise, just say the number.



Reasoning about software: A story

Let's play a game of *Fizzbuzz!*

- Take turns counting to 100.
- Number divisible by 3 (but not 5) $\Rightarrow$ say *Fizz!*
- Number divisible by 5 (but not 3) $\Rightarrow$ say *Buzz!*
- Number is divisible by 3 *and* 5 $\Rightarrow$ say *Fizzbuzz!*
- Otherwise, just say the number.



Reasoning about software: A story

```
def fizzbuzz1 =
  n := 1;
  while n ⩽ 100 do
    if 3|n then
      if not 5|n then
        print fizz; n++;
      else
        print fizzbuzz; n++;
    else if 5|n then
      print buzz; n++;
    else
      print n; n++;
  print done!;
```

Reasoning about software: A story

```
def fizzbuzz1 =
  n := 1;
  while n ⩽ 100 do
    if 3|n then
      if not 5|n then
        print fizz; n++;
      else
        print fizzbuzz; n++;
    else if 5|n then
      print buzz; n++;
    else
      print n; n++;
  print done!;
```

```
def fizzbuzz2 =
  n := 1;
  while n ⩽ 100 do
    if 5|n and 3|n then
      print fizzbuzz;
    else if 3|n then
      print fizz;
    else if 5|n then
      print buzz;
    else
      print n;
    n++;
  print done!;
```

Reasoning about software: A story

```
def fizzbuzz1 =
  n := 1;
  while n ⩽ 100 do
    if 3|n then
      if not 5|n then
        print fizz; n++;
      else
        print fizzbuzz; n++;
    else if 5|n then
      print buzz; n++;
    else
      print n; n++;
  print done!;
```

```
def fizzbuzz2 =
  n := 1;
  while n ⩽ 100 do
    if 5|n and 3|n then
      print fizzbuzz;
    else if 3|n then
      print fizz;
    else if 5|n then
      print buzz;
    else
      print n;
    n++;
  print done!;
```

$$\text{fizzbuzz1} \stackrel{?}{=} \text{fizzbuzz2}$$

Reasoning about software: A story

Starting with fizzbuzz1...

```
n := 1;
while n ⩽ 100 do
  if 3|n then
    if not 5|n then
      print fizz; n++;
    else
      print fizzbuzz; n++;
  else if 5|n then
    print buzz; n++;
  else
    print n; n++;
print done!;
```

Reasoning about software: A story

Move the n++; to the end...

```
n := 1;
while n ⩽ 100 do
   if 3|n then
      if not 5|n then
         print fizz;
      else
         print fizzbuzz;
   else if 5|n then
      print buzz;
   else
      print n;
    n++;
print done!;
```

Reasoning about software: A story

Negate not $5|n$ and flip the branches

```
n := 1;
while n ⩽ 100 do
    if 3|n then
        if 5|n then
            print fizzbuzz;
        else
            print fizz;
    else if 5|n then
        print buzz;
    else
        print n;
     n++;
print done!;
```

Reasoning about software: A story

Merge $3|n$ and $5|n$

```
n := 1;
while n ⩽ 100 do
  if 3|n and 5|n then
    print fizzbuzz;
  else if 3|n then
    print fizz;
  else if 5|n then
    print buzz;
  else
    print n;
  n++;
print done!;
```

Reasoning about software: A story

This is precisely fizzbuzz2!

```
n := 1;
while n ⩽ 100 do
  if 3|n and 5|n then
    print fizzbuzz;
  else if 3|n then
    print fizz;
  else if 5|n then
    print buzz;
  else
    print n;
  n++;
print done!;
```

Taking a step back

- The reasoning steps applied are very general. For instance:

$$\boxed{\text{if not b then e else f}} = \boxed{\text{if b then f else e}}$$

should work regardless of what b, e and f are.

Taking a step back

- The reasoning steps applied are very general. For instance:

  $$\boxed{\text{if not b then e else f}} = \boxed{\text{if b then f else e}}$$

  should work regardless of what b, e and f are.

- We treated the program as an expression, and reasoned equationally.

  > *programs are mathematical expressions, [...] subject to a set of laws as rich and elegant as those of any other branch of mathematics* (Hoare et al. 1984)

Taking a step back

By abstracting away from individual actions and tests, we go from . . .

```
def fizzbuzz1 =
  n := 1;
  while n ⩽ 100 do
    if 3|n then
      if not 5|n then
        print fizz; n++;
      else
        print fizzbuzz; n++;
    else if 5|n then
      print buzz; n++;
    else
      print n; n++;
  print done!;
```

```
def fizzbuzz2 =
  n := 1;
  while n ⩽ 100 do
    if 5|n and 3|n then
      print fizzbuzz;
    else if 3|n then
      print fizz;
    else if 5|n then
      print buzz;
    else
      print n;
    n++;
  print done!;
```

Taking a step back

...to *propositional* programs:

```
def fizzbuzz1 =
  p;
  while b do
    if c then
      if not d then
        r; u;
      else
        q; u;
    else if d then
      s; u;
    else
      t; u;
  v;
```

```
def fizzbuzz2 =
  p;
  while b do
    if d and c then
      q;
    else if c then
      r;
    else if d then
      s;
    else
      t;
    u;
  v;
```

Formalizing our reasoning

$$\boxed{\text{if } b \text{ then } e \text{ else } f} \quad = \quad \boxed{\text{if not } b \text{ then } f \text{ else } e} \qquad (\textit{skew commutativity})$$

Formalizing our reasoning

if b then e else f	=	if not b then f else e	(*skew commutativity*)
(if b then e else f);g	=	if b then e;g else f;g	(*left distributivity*)

Formalizing our reasoning

if b then e else f	=	if not b then f else e	(*skew commutativity*)
(if b then e else f);g	=	if b then e;g else f;g	(*left distributivity*)

if b then
(if c then e else f)
else
g
	=	if b and c then
		e
		else
		(if b then f else g)
			(*skew associativity*)

Formalizing our reasoning

if b then e else f	=	if not b then f else e	(*skew commutativity*)
(if b then e else f);g	=	if b then e;g else f;g	(*left distributivity*)

if b then
    (if c then e else f)
else
    g
	=	if b and c then
		    e
		else
		    (if b then f else g)
			(*skew associativity*)

while b do
    e
	=	if b then
		    e
		    while b do
		        e
			(*loop unrolling*)

Formalizing our reasoning: Kleene Algebra with Tests

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$\mathsf{b} \mid \mathsf{p} \mid e + f \mid e; f \mid e^*$$

- Extends regular expressions with a Boolean algebra of tests

$$\boxed{\text{if b then e else f}} = \mathsf{b}; \mathsf{e} + (\text{not b}); f$$

$$\boxed{\text{while b do e}} = (\mathsf{b}; \mathsf{e})^* ; (\text{not b})$$

(Kozen 1996), (Kozen & Smith 1996)

Formalizing our reasoning: Kleene Algebra with Tests

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e + f \mid e; f \mid e^*$$

- Extends regular expressions with a Boolean algebra of tests

$$\boxed{\text{if } b \text{ then } e \text{ else } f} = b; e + (\text{not } b); f$$

$$\boxed{\text{while } b \text{ do } e} = (b; e)^* ; (\text{not } b)$$

- Language semantics in terms of *guarded strings*: $\alpha_1 p_1 \alpha_2 p_2 \alpha_3 p_3 \cdots \alpha_n p_n \alpha_{n+1}$

(Kozen 1996), (Kozen & Smith 1996)

Formalizing our reasoning: Kleene Algebra with Tests

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e + f \mid e; f \mid e^*$$

- Extends regular expressions with a Boolean algebra of tests

$$\boxed{\text{if } b \text{ then e else f}} = b; e + (\text{not } b); f$$
$$\boxed{\text{while } b \text{ do e}} = (b; e)^* \,;\, (\text{not } b)$$

- Language semantics in terms of *guarded strings*: $\alpha_1 p_1 \alpha_2 p_2 \alpha_3 p_3 \cdots \alpha_n p_n \alpha_{n+1}$
- Complete and finitary axiomatization

(Kozen 1996), (Kozen & Smith 1996)

Formalizing our reasoning: Kleene Algebra with Tests

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e + f \mid e; f \mid e^*$$

- Extends regular expressions with a Boolean algebra of tests

$$\boxed{\text{if } b \text{ then } e \text{ else } f} = b; e + (\text{not } b); f$$

$$\boxed{\text{while } b \text{ do } e} = (b; e)^* \,; (\text{not } b)$$

- Language semantics in terms of *guarded strings*: $\alpha_1 p_1 \alpha_2 p_2 \alpha_3 p_3 \cdots \alpha_n p_n \alpha_{n+1}$
- Complete and finitary axiomatization
- Non-determinism makes equivalence PSPACE-complete

(Kozen 1996), (Kozen & Smith 1996)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

The part of KAT specifically for while programs!

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

The part of KAT specifically for while programs!

$e +_b f =$ if b then e else f

$e^{(b)} =$ while b do e

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

The part of KAT specifically for while programs!

$$\begin{aligned} e +_b f &= \text{if } b \text{ then e else f} \\ &= b; e + (\text{not } b); f \end{aligned}$$

$$\begin{aligned} e^{(b)} &= \text{while } b \text{ do e} \\ &= (b; e)^* ; (\text{not } b) \end{aligned}$$

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

The part of KAT specifically for while programs!

$$e +_b f = \text{if } b \text{ then e else f} = b; e + (\text{not } b); f$$

$$e^{(b)} = \text{while } b \text{ do e} = (b; e)^* ; (\text{not } b)$$

$$\boxed{\text{fizzbuzz1}} = p; ((r; u +_{\text{not } d} q; u) +_c (s; u +_d t; u))^{(b)}; v$$

$$\boxed{\text{fizzbuzz2}} = p; ((q +_{d \text{ and } c} (r +_c (s +_d t)))u)^{(b)}; v$$

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_{b} f \mid e; f \mid e^{(b)}$$

- Language semantics in terms of *guarded strings*

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

- Language semantics in terms of *guarded strings*
- Language equivalence is efficiently decidable! (nearly linear)

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Formalizing our reasoning: Guarded KAT

Fix a set $\{p, q, \ldots\}$ of actions and a Boolean algebra $\{b, c, \ldots\}$ of tests

$$b \mid p \mid e +_b f \mid e; f \mid e^{(b)}$$

- Language semantics in terms of *guarded strings*
- Language equivalence is efficiently decidable! (nearly linear)
- Complete but infinitary axiomatization

(Kozen & Tseng 2008), (Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$\mathsf{0}; e = e; \mathsf{0} = \mathsf{0}$$

$$\mathsf{1}; e = e; \mathsf{1} = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$0; e = e; 0 = 0$$

$$1; e = e; 1 = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

fizzbuzz1

$$= \mathsf{p}; ((\mathsf{q}; \mathsf{u} +_{\mathsf{not\ d}} \mathsf{r}; \mathsf{u}) +_{\mathsf{c}} (\mathsf{s}; \mathsf{u} +_{\mathsf{d}} \mathsf{t}; \mathsf{u}))^{(\mathsf{b})}; \mathsf{v}$$

$= \text{fizzbuzz2}$

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$0; e = e; 0 = 0$$

$$1; e = e; 1 = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

fizzbuzz1

$$= \mathsf{p}; ((\mathsf{q}; \mathsf{u} +_{\mathsf{not\ d}} \mathsf{r}; \mathsf{u}) +_{\mathsf{c}} (\mathsf{s}; \mathsf{u} +_{\mathsf{d}} \mathsf{t}; \mathsf{u}))^{(\mathsf{b})}; \mathsf{v}$$

$$= \mathsf{p}; (((\mathsf{q} +_{\mathsf{not\ d}} \mathsf{r}) +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t})); \mathsf{u})^{(\mathsf{b})}; \mathsf{v}$$

= fizzbuzz2

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{b} e = e$$

$$e +_{b} f = f +_{\text{not } b} e$$

$$(e +_{b} f) +_{c} g = e +_{b \text{ and } c} (f +_{c} g)$$

$$e +_{b} f = b; e +_{b} f$$

Composition

$$b; c = b \text{ and } c$$

$$0; e = e; 0 = 0$$

$$1; e = e; 1 = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{b} f); g = e; g +_{b} f; g$$

fizzbuzz1

$$= p; ((q; u +_{\text{not } d} r; u) +_{c} (s; u +_{d} t; u))^{(b)}; v$$

$$= p; (((q +_{\text{not } d} r) +_{c} (s +_{d} t)); u)^{(b)}; v$$

$$= p; (((r +_{d} q) +_{c} (s +_{d} t)); u)^{(b)}; v$$

$= \text{fizzbuzz2}$

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$0; e = e; 0 = 0$$

$$1; e = e; 1 = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

fizzbuzz1

$$= \mathsf{p}; ((\mathsf{q}; \mathsf{u} +_{\mathsf{not\ d}} \mathsf{r}; \mathsf{u}) +_{\mathsf{c}} (\mathsf{s}; \mathsf{u} +_{\mathsf{d}} \mathsf{t}; \mathsf{u}))^{(\mathsf{b})}; \mathsf{v}$$

$$= \mathsf{p}; (((\mathsf{q} +_{\mathsf{not\ d}} \mathsf{r}) +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t})); \mathsf{u})^{(\mathsf{b})}; \mathsf{v}$$

$$= \mathsf{p}; (((\mathsf{r} +_{\mathsf{d}} \mathsf{q}) +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t})); \mathsf{u})^{(\mathsf{b})}; \mathsf{v}$$

$$= \mathsf{p}; ((\mathsf{r} +_{\mathsf{d\ and\ c}} (\mathsf{q} +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t}))); \mathsf{u})^{(\mathsf{b})}; \mathsf{v}$$

= fizzbuzz2

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$\mathsf{0}; e = e; \mathsf{0} = \mathsf{0}$$

$$\mathsf{1}; e = e; \mathsf{1} = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

Loops

$$e^{(\mathsf{b})}; f = e; (e^{(\mathsf{b})}; f) +_{\mathsf{b}} f$$

$$\frac{g = e; g +_{\mathsf{b}} f \quad e \text{ productive}}{g = e^{(\mathsf{b})}; f}$$

. . . and generalizations of the above

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$\mathsf{0}; e = e; \mathsf{0} = \mathsf{0}$$

$$\mathsf{1}; e = e; \mathsf{1} = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

Loops

$$e^{(\mathsf{b})}; f = e; (e^{(\mathsf{b})}; f) +_{\mathsf{b}} f$$

$$\frac{g = e; g +_{\mathsf{b}} f \quad e \text{ productive}}{g = e^{(\mathsf{b})}; f}$$

. . . and generalizations of the above

Open Question #1

Do we need the generalized loop rules?

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Axiomatizing our reasoning

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

$$e +_{\mathsf{b}} f = \mathsf{b}; e +_{\mathsf{b}} f$$

Composition

$$\mathsf{b}; \mathsf{c} = \mathsf{b\ and\ c}$$

$$\mathsf{0}; e = e; \mathsf{0} = \mathsf{0}$$

$$\mathsf{1}; e = e; \mathsf{1} = e$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

Loops

$$e^{(\mathsf{b})}; f = e; (e^{(\mathsf{b})}; f) +_{\mathsf{b}} f$$

$$\frac{g = e; g +_{\mathsf{b}} f \quad e \text{ productive}}{g = e^{(\mathsf{b})}; f}$$

...and generalizations of the above

Open Question #1

Do we need the generalized loop rules?

Open Question #2

Can we factor out the side condition?

(Smolka, Foster, Hsu, K., Kozen & Silva 2019)

Why is GKAT so hard?

Complete algebraic axiomatization of KAT goes back to Kozen (1996)...

Why is GKAT so hard?

Complete algebraic axiomatization of KAT goes back to Kozen (1996)...

GKAT programs are a proper subset of KAT programs...

Why is GKAT so hard?

Complete algebraic axiomatization of KAT goes back to Kozen (1996)...

GKAT programs are a proper subset of KAT programs...

So why is GKAT so dificult?

Kleene Algebra with Tests

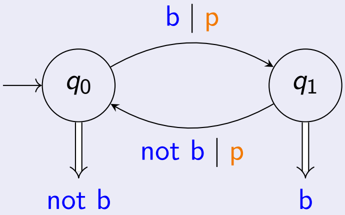
Not every deterministic KAT program is a GKAT program.

Kleene Algebra with Tests

Not every deterministic KAT program is a GKAT program.

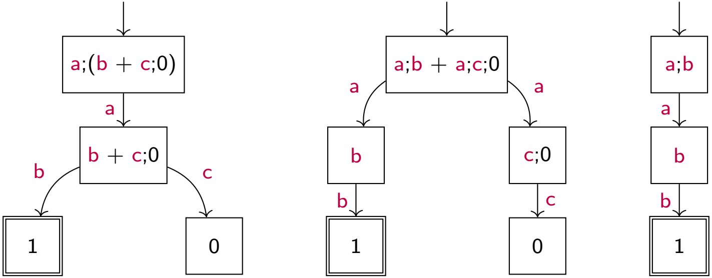
Example (Schmid, K., Kozen & Silva 2021), (Kozen & Tseng 2008)

```
while b do
    p;
    if b then break;
    p
```



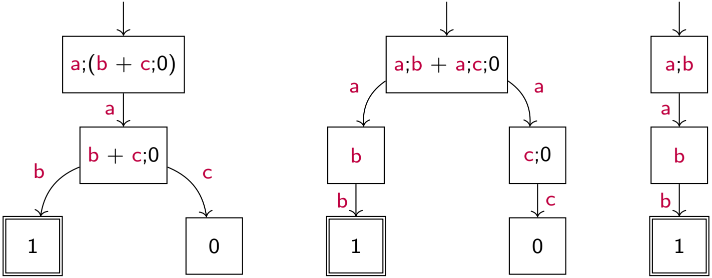
A similar problem in process algebra

Milner studied *regular expressions up to bisimilarity* in 1984.



A similar problem in process algebra

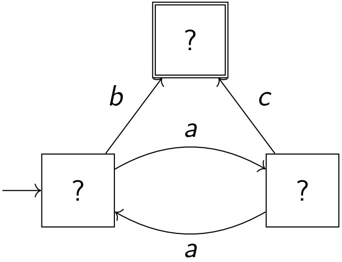
Milner studied *regular expressions up to bisimilarity* in 1984.



He proposed axioms for equivalence, but left completeness open.

A similar problem in process algebra

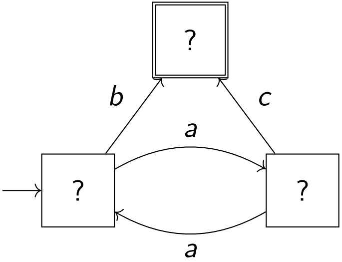
Not all behaviors realized as expressions (Milner 1984), (Bosscher 1997)



$$\mu x(b + a \cdot (c + a \cdot x))$$

A similar problem in process algebra

Not all behaviors realized as expressions (Milner 1984), (Bosscher 1997)



$$\mu x(b + a \cdot (c + a \cdot x))$$

Axiomatization for fragment *without 1* (Grabmayer & Fokkink 2020)

$$0 \mid a \mid e + f \mid e; f \mid e^* f$$

Introducing. . . Skip-free GKAT!

The *skip-free fragment* of GKAT is given by

$$0 \mid p \mid e +_{b} f \mid e \,;\, f \mid e^{(b)} f$$

Introducing... Skip-free GKAT!

The *skip-free fragment* of GKAT is given by

$$0 \mid \mathsf{p} \mid e +_{\mathsf{b}} f \mid e \,;\, f \mid e^{(\mathsf{b})} f$$

- Can still express a wide range of programs...

Introducing...Skip-free GKAT!

The *skip-free fragment* of GKAT is given by

$$0 \mid \mathsf{p} \mid e +_{\mathsf{b}} f \mid e \mathbin{;} f \mid e^{(\mathsf{b})} f$$

- Can still express a wide range of programs...

$$\boxed{\text{fizzbuzz1}} = \mathsf{p}; ((\mathsf{r}; \mathsf{u} +_{\mathsf{not\ d}} \mathsf{q}; \mathsf{u}) +_{\mathsf{c}} (\mathsf{s}; \mathsf{u} +_{\mathsf{d}} \mathsf{t}; \mathsf{u}))^{(\mathsf{b})} \mathsf{v}$$

$$\boxed{\text{fizzbuzz2}} = \mathsf{p}; ((\mathsf{q} +_{\mathsf{d\ and\ c}} (\mathsf{r} +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t})))\mathsf{u})^{(\mathsf{b})} \mathsf{v}$$

Introducing... Skip-free GKAT!

The *skip-free fragment* of GKAT is given by

$$0 \mid \mathsf{p} \mid e +_{\mathsf{b}} f \mid e \,;\, f \mid e^{(\mathsf{b})} f$$

- Can still express a wide range of programs...

$$\boxed{\text{fizzbuzz1}} = \mathsf{p}; ((\mathsf{r}; \mathsf{u} +_{\mathsf{not\ d}} \mathsf{q}; \mathsf{u}) +_{\mathsf{c}} (\mathsf{s}; \mathsf{u} +_{\mathsf{d}} \mathsf{t}; \mathsf{u}))^{(\mathsf{b})} \mathsf{v}$$

$$\boxed{\text{fizzbuzz2}} = \mathsf{p}; ((\mathsf{q} +_{\mathsf{d\ and\ c}} (\mathsf{r} +_{\mathsf{c}} (\mathsf{s} +_{\mathsf{d}} \mathsf{t})))\mathsf{u})^{(\mathsf{b})} \mathsf{v}$$

- Every skip-free expression satisfies the side condition!

$$\frac{g = e; g +_{\mathsf{b}} f \quad e \text{ productive}}{g = e^{(\mathsf{b})}; f} \implies \frac{g = e; g +_{\mathsf{b}} f}{g = e^{(\mathsf{b})} f}$$

Axioms for Skip-free GKAT

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

Composition

$$0; e = e; 0 = 0$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

Loops

$$e^{(\mathsf{b})} f = e; (e^{(\mathsf{b})} f) +_{\mathsf{b}} f$$

$$\frac{g = e; g +_{\mathsf{b}} f}{g = e^{(\mathsf{b})} f}$$

Axioms for Skip-free GKAT

Conditionals

$$e +_{\mathsf{b}} e = e$$

$$e +_{\mathsf{b}} f = f +_{\mathsf{not\ b}} e$$

$$(e +_{\mathsf{b}} f) +_{\mathsf{c}} g = e +_{\mathsf{b\ and\ c}} (f +_{\mathsf{c}} g)$$

Composition

$$0; e = e; 0 = 0$$

$$e; (f; g) = (e; f); g$$

$$(e +_{\mathsf{b}} f); g = e; g +_{\mathsf{b}} f; g$$

Loops

$$e^{(\mathsf{b})} f = e; (e^{(\mathsf{b})} f) +_{\mathsf{b}} f$$

$$\frac{g = e; g +_{\mathsf{b}} f}{g = e^{(\mathsf{b})} f}$$

Completeness Theorem

(K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Axioms for Skip-free GKAT

Conditionals

$$e +_{b} e = e$$

$$e +_{b} f = f +_{\text{not } b} e$$

$$(e +_{b} f) +_{c} g = e +_{b \text{ and } c} (f +_{c} g)$$

Composition

$$0; e = e; 0 = 0$$

$$e; (f; g) = (e; f); g$$

$$(e +_{b} f); g = e; g +_{b} f; g$$

Loops

$$e^{(b)} f = e; (e^{(b)} f) +_{b} f$$

$$\frac{g = e; g +_{b} f}{g = e^{(b)} f}$$

Completeness Theorem

(K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

This is an *algebraic* and *finitary* axiomatization!

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Proof sketch.

We designed two transformations:

- gtr: 1-free GKAT expressions $\rightarrow$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\rightarrow$ 1-free GKAT expressions

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Proof sketch.

We designed two transformations:

- gtr: 1-free GKAT expressions $\to$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\to$ 1-free GKAT expressions

Given 1-free GKAT expressions e and f with $[\![e]\!] = [\![f]\!]$:

$$[\![gtr(e)]\!] = [\![gtr(f)]\!]$$

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Proof sketch.

We designed two transformations:

- gtr: 1-free GKAT expressions $\to$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\to$ 1-free GKAT expressions

Given 1-free GKAT expressions e and f with $[\![e]\!] = [\![f]\!]$:

$$[\![gtr(e)]\!] = [\![gtr(f)]\!] \implies gtr(e) \equiv gtr(f)$$

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Proof sketch.

We designed two transformations:

- gtr: 1-free GKAT expressions $\to$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\to$ 1-free GKAT expressions

Given 1-free GKAT expressions e and f with $[\![e]\!] = [\![f]\!]$:

$$[\![gtr(e)]\!] = [\![gtr(f)]\!] \implies gtr(e) \equiv gtr(f) \implies rtg(gtr(e)) \equiv rtg(gtr(f))$$

Axioms for Skip-free GKAT

Completeness Theorem (K., Schmid & Silva 2023)

For e, f skip-free GKAT expressions, the following are equivalent:

1. e and f are language equivalent
2. the equation $e = f$ is provable

Proof sketch.

We designed two transformations:

- gtr: 1-free GKAT expressions $\to$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\to$ 1-free GKAT expressions

Given 1-free GKAT expressions e and f with $[\![e]\!] = [\![f]\!]$:

$$[\![gtr(e)]\!] = [\![gtr(f)]\!] \implies gtr(e) \equiv gtr(f) \implies rtg(gtr(e)) \equiv rtg(gtr(f)) \implies e \equiv f \qquad \square$$

Future work

Regex/bisimilarity

(Milner 1984)

Future work

Regex/bisimilarity

(Milner 1984)

⋮

Completeness of
1-free regex/bisimilarity

(Grabmayer & Fokkink 2020)

Future work

Regex/bisimilarity
(Milner 1984)

⋮

Completeness of
1-free regex/bisimilarity
(Grabmayer & Fokkink 2020)

While fragment of KAT
(Kozen & Tseng 2008)

Future work

Regex/bisimilarity
(Milner 1984)

⋮

Completeness of
1-free regex/bisimilarity
(Grabmayer & Fokkink 2020)

While fragment of KAT
(Kozen & Tseng 2008)

GKAT
(Smolka et al, 2019)

Future work

Regex/bisimilarity

(Milner 1984)

⋮

Completeness of
1-free regex/bisimilarity

(Grabmayer & Fokkink 2020)

While fragment of KAT

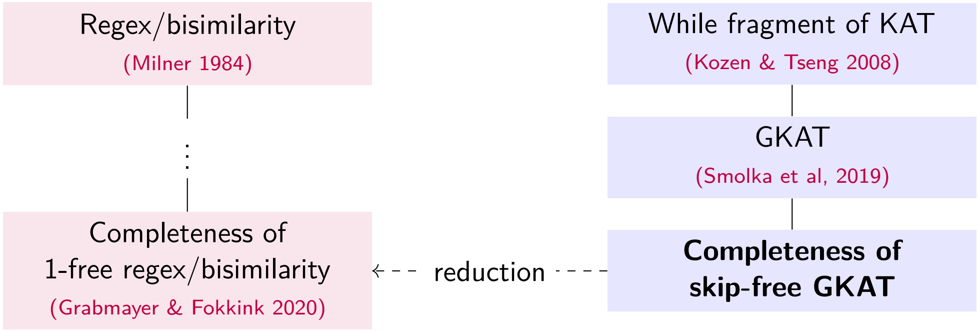
(Kozen & Tseng 2008)

GKAT

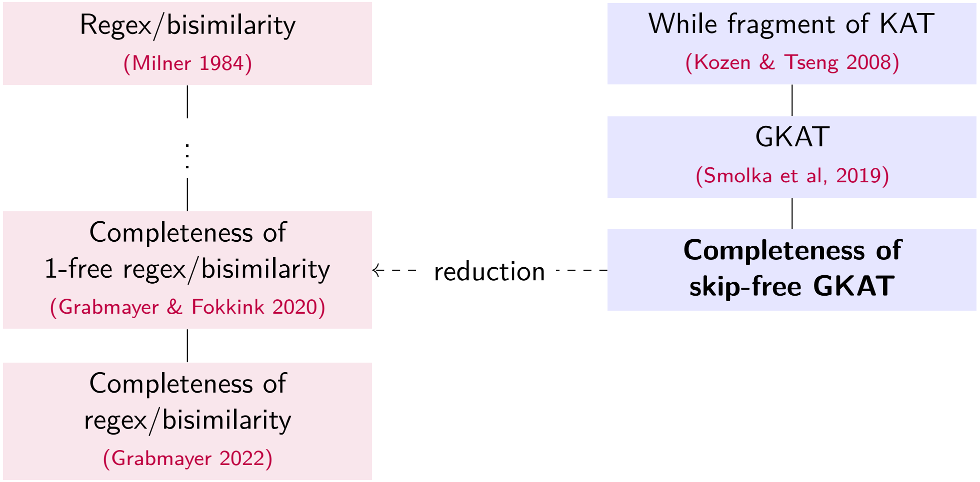
(Smolka et al, 2019)

**Completeness of
skip-free GKAT**

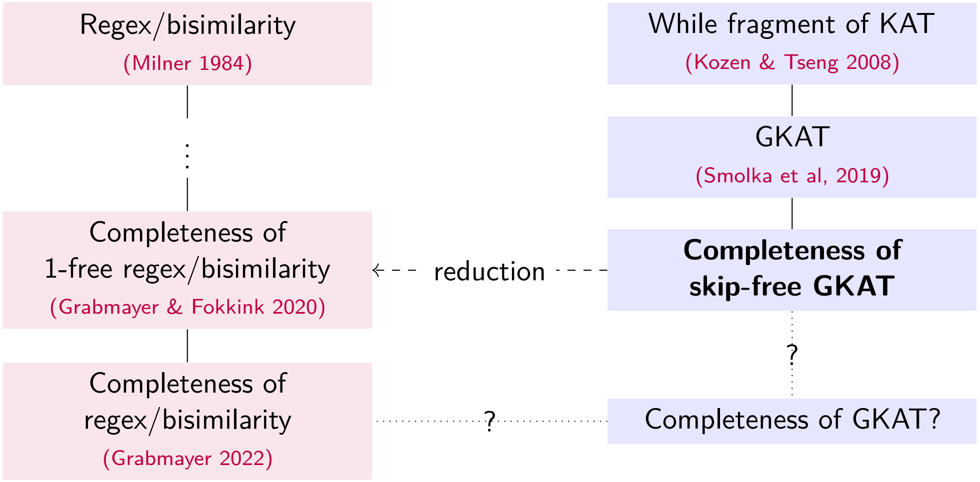
Future work


Regex/bisimilarity
(Milner 1984)
Completeness of
1-free regex/bisimilarity
(Grabmayer & Fokkink 2020)
reduction
While fragment of KAT
(Kozen & Tseng 2008)
GKAT
(Smolka et al, 2019)
Completeness of
skip-free GKAT

Future work


Regex/bisimilarity
(Milner 1984)
Completeness of
1-free regex/bisimilarity
(Grabmayer & Fokkink 2020)
Completeness of
regex/bisimilarity
(Grabmayer 2022)
While fragment of KAT
(Kozen & Tseng 2008)
GKAT
(Smolka et al, 2019)
Completeness of
skip-free GKAT
reduction

Future work



Regex/bisimilarity
(Milner 1984)
Completeness of
1-free regex/bisimilarity
(Grabmayer & Fokkink 2020)
Completeness of
regex/bisimilarity
(Grabmayer 2022)
While fragment of KAT
(Kozen & Tseng 2008)
GKAT
(Smolka et al, 2019)
Completeness of
skip-free GKAT
reduction
?
Completeness of GKAT?
?

Recap

- GKAT: Propositional while programs/language equivalence (Smolka et al. 2019)
- **Open problem:** Is finite axiomatization of GKAT complete?
- Similar problem in process algebra (Milner 1984) — open for 38 years!
- Inspired by (Grabmayer & Fokkink 2020) we introduce skip-free GKAT

$$0 \mid a \mid e + f \mid e; f \mid e^* f \qquad \Longrightarrow \qquad 0 \mid \mathsf{p} \mid e +_{\mathsf{b}} f \mid e; f \mid e^{(\mathsf{b})} f$$

- **Theorem:** Finite axiomatization is complete for skip-free GKAT
  - Completeness proof is a reduction to (Grabmayer & Fokkink 2020)
- **New question:** Can we reduce all of GKAT to regex/bisimilarity?

Questions are welcome!