On propositonal program equivalence

Tobias Kappé

WoLLIC 2025

Universiteit Leiden
The Netherlands

liacs Leiden Institute of Advanced Computer Science

Program equivalence

```
if x = 1 then
  │ y++;
else
  │ z := z/2;
end
```

$\equiv$

```
if x ≠ 1 then
  │ z := z/2;
else
  │ y++;
end
```

Propositional program equivalence

```
if b then
|  p;
else
|  q;
end
```

$\equiv$

```
if not b then
|  q;
else
|  p;
end
```

Program equivalence

```
s := new stack();
node := root;
while node != nil do
    s.push(node);
    node := node.left;
end
while !s.empty do
    node = s.pop();
    visit(node);
    node = node.right;
    while node != nil do
        s.push(node);
        node := node.left;
    end
end
```

$\equiv$

```
s := new stack();
node := root;
while node != nil or !s.empty do
    if node != nil then
        s.push(node);
        node = node.left;
    else
        node = s.pop();
        visit(node);
        node = node.right;
    end
end
```

(courtesy of Hendrik Jan Hoogeboom)

Propositional program equivalence

```
s;
while b do
  p;
end
while c do
  q;
  while b do
    p;
  end
end
```

$\equiv$

```
s;
while b or c do
  if b then
    p;
  else
    q;
  end
end
```

Kleene Algebra with Tests (KAT)

(Kozen 1996), (Kozen & Smith 1996)

Fix a set $\Sigma = \{\mathsf{p}, \mathsf{q}, \ldots\}$ of actions and a set $T = \{\mathsf{s}, \mathsf{t}, \ldots\}$ of tests

$$\mathsf{BExp} \ni \mathsf{b}, \mathsf{c} ::= \mathsf{t} \in T \mid \mathsf{b} + \mathsf{c} \mid \mathsf{b} \cdot \mathsf{c} \mid \overline{\mathsf{b}}$$
$$\mathsf{KExp} \ni \mathsf{e}, \mathsf{f} ::= \mathsf{b} \in \mathsf{BExp} \mid \mathsf{p} \in \Sigma \mid \mathsf{e} + \mathsf{f} \mid \mathsf{e} \cdot \mathsf{f} \mid \mathsf{e}^*$$

- Language semantics in terms of *guarded strings*:

$$\alpha_1 \mathsf{p}_1 \alpha_2 \mathsf{p}_2 \alpha_3 \mathsf{p}_3 \cdots \alpha_n \mathsf{p}_n \alpha_{n+1}$$

where $\alpha_i \in 2^T$ and $\mathsf{p}_i \in \Sigma$. For example:

$$[\![\mathsf{t}]\!] = \{\alpha \in 2^T : \mathsf{t} \in \alpha\} \qquad [\![\mathsf{e} + \mathsf{f}]\!] = [\![\mathsf{e}]\!] \cup [\![\mathsf{f}]\!] \qquad [\![\mathsf{e} \cdot \mathsf{f}]\!] = \{w\alpha x : w\alpha \in [\![\mathsf{e}]\!], \alpha x \in [\![\mathsf{f}]\!]\}$$

- We can embed simple propositional programs:

$$\text{if } \mathsf{b} \text{ then } \mathsf{e} \text{ else } \mathsf{f} \mapsto \mathsf{b} \cdot \mathsf{e} + \overline{\mathsf{b}} \cdot \mathsf{f}$$
$$\text{while } \mathsf{b} \text{ do } \mathsf{e} \mapsto (\mathsf{b} \cdot \mathsf{e})^* \cdot \overline{\mathsf{b}}$$

Kleene Algebra with Tests (KAT), cont'd

(Kozen 1996), (Kozen & Smith 1996)

Fix a set $\Sigma = \{\mathsf{p}, \mathsf{q}, \ldots\}$ of actions and a set $T = \{\mathsf{s}, \mathsf{t}, \ldots\}$ of tests

$$\mathsf{BExp} \ni \mathsf{b}, \mathsf{c} ::= \mathsf{t} \in T \mid \mathsf{b} + \mathsf{c} \mid \mathsf{b} \cdot \mathsf{c} \mid \overline{\mathsf{b}}$$
$$\mathsf{KExp} \ni \mathsf{e}, \mathsf{f} ::= \mathsf{b} \in \mathsf{BExp} \mid \mathsf{p} \in \Sigma \mid \mathsf{e} + \mathsf{f} \mid \mathsf{e} \cdot \mathsf{f} \mid \mathsf{e}^*$$

- Complete, quasi-equational axiomatization of equivalence.
- Equivalence is decidable (via automata), but PSPACE-hard.
- However: in practice, this scales rather well!
- Most practical programs encoded in KAT are deterministic.

Guarded KAT (GKAT)

(Kozen & Tseng 2008), (Smolka et al. 2020)

Fix a set $\Sigma = \{\mathsf{p}, \mathsf{q}, \ldots\}$ of actions and a set $T = \{\mathsf{s}, \mathsf{t}, \ldots\}$ of tests

$$\mathsf{BExp} \ni \mathsf{b}, \mathsf{c} ::= \mathsf{t} \in T \mid \mathsf{b} + \mathsf{c} \mid \mathsf{b} \cdot \mathsf{c} \mid \overline{\mathsf{b}}$$

$$\mathsf{GExp} \ni \mathsf{e}, \mathsf{f} ::= \texttt{assert } \mathsf{b} \mid \mathsf{p} \in \Sigma \mid \mathsf{e} \cdot \mathsf{f} \mid \texttt{if } \mathsf{b} \texttt{ then } \mathsf{e} \texttt{ else } \mathsf{f} \mid \texttt{while } \mathsf{b} \texttt{ do } \mathsf{e}$$

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

GKAT semantics

By example

$$\left[\!\!\left[\begin{array}{l}\texttt{if } t \texttt{ then}\\ \quad\mid\quad p;\\ \texttt{else}\\ \quad\mid\quad q;\\ \texttt{end}\end{array}\right]\!\!\right] = \left\{\begin{array}{l} tpt,\\ tp\bar{t},\\ \bar{t}q\bar{t},\\ \bar{t}qt \end{array}\right\}$$

GKAT semantics

By example

$$
\left\llbracket
\begin{array}{l}
\texttt{while } \mathsf{t} \texttt{ or } \mathsf{s} \texttt{ do} \\
\quad \texttt{if } \mathsf{t} \texttt{ then} \\
\qquad \mathsf{p}; \\
\quad \texttt{else} \\
\qquad \mathsf{q}; \\
\quad \texttt{end} \\
\texttt{end}
\end{array}
\right\rrbracket
=
\left\{
\begin{array}{l}
\overline{\mathsf{t}}\,\overline{\mathsf{s}}, \\
\overline{\mathsf{t}}\mathsf{s}\mathsf{q}\overline{\mathsf{t}}\,\overline{\mathsf{s}}, \\
\mathsf{t}\overline{\mathsf{s}}\mathsf{p}\overline{\mathsf{t}}\mathsf{s}\mathsf{q}\overline{\mathsf{t}}\,\overline{\mathsf{s}}, \\
\dots
\end{array}
\right\}
$$

GKAT decision procedure

```
while t or s do
  if t then
    p;
  else
    q;
  end
end
```

$\mapsto$

$q_0 \Longrightarrow \bar{t}\bar{s}$

$ts \mid p$

$t\bar{s} \mid p$

$\bar{t}s \mid q$

See (Smolka et al. 2020) for the full details.

GKAT decision procedure, cont'd

Theorem (Smolka et al. '20)

For every GKAT expression e*, we can construct a GKAT automaton* A_{e} *such that* $[\![\mathsf{e}]\!] = [\![A_{\mathsf{e}}]\!]$.

This is a basic syntax-directed translation, à la (Thompson 1968).

Lemma

A_{e} *is at most linear in the size of* e.

Equivalence (bisimilarity) of GKAT automata can be easily decided.

GKAT axiomatization

Are there rules that characterize when $[\![e]\!] = [\![f]\!]$?

We will use the condensed syntax:

$$\mathrm{GExp} \ni e, f ::= b \in \mathrm{BExp} \mid p \in \Sigma \mid e \cdot f \mid e +_b f \mid e^{(b)}$$

Order of operations: first $-^{(b)}$, then $\cdot$, then $+_b$.

GKAT axiomatization, cont'd

Conditionals

$$e +_b e = e$$

$$e +_b f = f +_{\bar{b}} e$$

$$(e +_b f) +_c g = e +_{b \cdot c} (f +_c g)$$

$$e +_b f = b \cdot e +_b f$$

Composition

$$0 \cdot e = e \cdot 0 = 0$$

$$1 \cdot e = e \cdot 1 = e$$

$$e \cdot (f \cdot g) = (e \cdot f) \cdot g$$

$$(e +_b f) \cdot g = e \cdot g +_b f \cdot g$$

Loops

$$e^{(b)} \cdot f = e \cdot (e^{(b)} \cdot f) +_b f$$

$$(e +_b 1)^{(c)} = (b \cdot e)^{(c)}$$

$$\frac{g = e \cdot g +_b f \quad e \text{ productive}}{g = e^{(b)} \cdot f}$$

...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?

Theorem (Smolka et al. '20)

For all GKAT expressions e and f, we have $[\![e]\!] = [\![f]\!]$ if and only if $e = f$ follows from the rules above.

GKAT axiomatization, cont'd

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

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

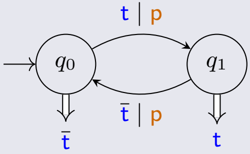
So what makes GKAT so dificult?

GKAT expressivity

Every finite automaton corresponds to a regular expression...

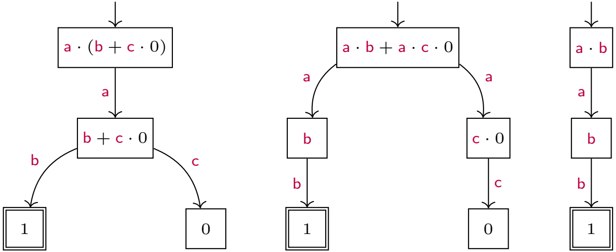
...but not every GKAT automaton corresponds to a GKAT program!

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



A detour through process algebra

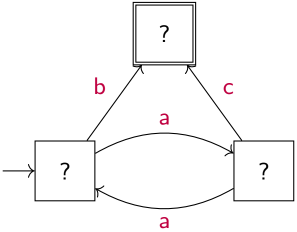
Milner studied *regular expressions up to bisimilarity* in 1984.



He proposed axioms for equivalence, but left completeness open.

A detour through process algebra

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



But there is an axiomatization for fragment *without 1* (Grabmayer & Fokkink 2020)

$$0 \mid \mathsf{a} \mid e + f \mid e \cdot f \mid e^* f$$

Skip-free GKAT

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

$$\mathrm{GExp}^- \ni e, f ::= 0 \mid p \mid e +_b f \mid e \cdot f \mid e^{(b)} f$$

- Every skip-free expression satisfies the side condition!

$$\frac{g = e \cdot g +_b f \quad e \text{ productive}}{g = e^{(b)} \cdot f} \qquad \Longrightarrow \qquad \frac{g = e \cdot g +_b f}{g = e^{(b)} f}$$

Skip-free GKAT axiomatization

Conditionals

$$e +_b e = e$$

$$e +_b f = f +_{\overline{b}} e$$

$$(e +_b f) +_c g = e +_{b \cdot c} (f +_c g)$$

Composition

$$0 \cdot e = e \cdot 0 = 0$$

$$e \cdot (f \cdot g) = (e \cdot f) \cdot g$$

$$(e +_b f) \cdot g = e \cdot g +_b f \cdot g$$

Loops

$$e^{(b)} f = e \cdot (e^{(b)} f) +_b f$$

$$\frac{g = e \cdot g +_b f}{g = e^{(b)} f}$$

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. $e = f$ is provable from these axioms

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

Skip-free GKAT axiomatization

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. $\mathsf{e} = \mathsf{f}$ is provable from the axioms

Proof sketch.

We designed two transformations:

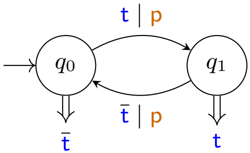
- gtr: Skip-free GKAT expressions $\to$ fragment of 1-free regex
- rtg: fragment of 1-free regex $\to$ skip-free GKAT expressions

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

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

More control flow operators

Remember this automaton?

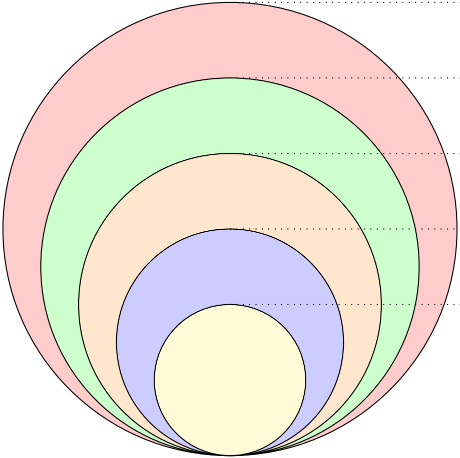


A pseudo-expression for this could be

$$\texttt{if } t \texttt{ then } \{\texttt{repeat } p \texttt{ while } t \texttt{ changes}\} \texttt{ else assert } 1$$

What if we add `repeat` e `while` b `changes` as an operator to GKAT?

A hierarchy



KAT: $e \cdot f$, $e + f$, e^*

deterministic KAT

GKAT + `repeat` e `while` b `changes` + ???

GKAT + `repeat` e `while` b `changes`

GKAT: $e \cdot f$, `if` b `then` e `else` f, `while` b `do` e

A hierarchy

Theorem (Ten Cate & K. '25)

For any deterministic fragment of KAT *with finitely many operators* ($e \cdot f$, `if` b `then` e `else` f, `while` b `do` e, ...), *there exist a deterministic* KAT *expression outside this fragment.*

Goto removal

```
L:
if t then
    p;
    goto R;
return;
R:
if not t then
    q;
    goto L;
return;
```

goto removal
(e.g., calipso)
→

```
x := 1;
while x ≠ 0 do
    if x = 1 and t
     then
        p;
        x := 2;
    else if x = 2 and
     not t then
        q;
        x := 1;
    else
        x := 0;
end
```

See (Erosa & Hendren '94; Cassé et al. '02)

(De)compilation

```
L:
if t then
    p;
    goto R;
return;
R:
if not t then
    q;
    goto L;
return;
```

GCC ⟶ executable ⟶ GHIDRA

```
while t do
    p;
    if t then
        break;
    q;
end
```

Enter CF-GKAT

Fix sets of:

- *actions* $\mathsf{p} \in \Sigma$;
- *tests* $\mathsf{b} \in T$;
- *labels* $\ell \in L$; and
- *indicator values* $i \in I$.

CF-GKAT is GKAT augmented with non-local flow control:

$$\begin{aligned}\mathrm{GExp} \ni \mathsf{e}, \mathsf{f} ::= {} & \texttt{assert } \mathsf{b} \mid \mathsf{p} \in \Sigma \mid x := i \ (i \in I) \mid \mathsf{e}; \mathsf{f} \mid \texttt{if } \mathsf{b} \texttt{ then } \mathsf{e} \texttt{ else } \mathsf{f} \mid \\ & \texttt{while } \mathsf{b} \texttt{ do } \mathsf{e} \mid \texttt{break} \mid \texttt{return} \mid \texttt{goto } \ell \ (\ell \in L) \mid \texttt{label } \ell \ (\ell \in L)\end{aligned}$$

See also (Kozen '08) for how to do this in KAT.

CF-GKAT semantics

By example

```
while t do
    p;
    if t then
        break;
    q;
end
```

$$= \left\{ \begin{array}{l} t p \bar{t} q \bar{t}, \\ t p \bar{t} q t p \bar{t} q \bar{t}, \\ t p t \\ t p \bar{t} q t p t \\ \ldots \end{array} \right\}$$

```
L:
if t then
    p;
    goto R;
return;
R:
if not t then
    q;
    goto L;
return;
```

$$= \left\{ \begin{array}{l} t p t \\ t p \bar{t} q \bar{t}, \\ t p \bar{t} q t p \bar{t} q \bar{t}, \\ t p \bar{t} q t p t \\ \ldots \end{array} \right\}$$

Decision procedure

CF-GKAT automata

```
if t and x = 1 then
  x := 2;
  L:
  p;
else
  x := 1;
  goto L;
```

$\mapsto$

$$
\begin{tikzcd}
(q_0, 2) \arrow[drr, bend left, "t / p \;\; \bar{t} / p"] & & \\
(q_0, 1) \arrow[rr, "\bar{t} \mid p"] \arrow[d, "t \mid p"] & & (q_1, 1) \arrow[d, Rightarrow] \\
(q_1, 2) \arrow[rr, Rightarrow] & & t, \bar{t}
\end{tikzcd}
$$

Decision procedure

Translating to CF-GKAT automata

Theorem (Zhang, K., Narváez & Naus '25)

For every CF-GKAT expression e*, we can construct a GKAT automaton* A_{e} *such that* $[\![\mathsf{e}]\!] = [\![A_{\mathsf{e}}]\!]$.

Another syntax-directed translation, à la (Thompson 1968).

Lemma

A_{e} *is linear in the size of* e *and* $|I|$.

ROCQ

Validation

Blinding

C programs are still not CF-GKAT programs:

- Parsing C can be really hard!
- Same statements mapped to same variable.
- Which variables are indicators?
- Macros may hide relevant information.

LLVM/clang to the rescue

Validation

Blinding

```
#define hidden(x) \
    x=f(x); goto R;

int foo(int x, int y) {
  L:
  if (x == y) {
    x = f(x);
    hidden(x);
  }
  return;
  R:
  if (x != y) {
    y = g(y);
    goto L;
  }
}
```

$\mapsto$

```
L:
if t then
  | p;
  | goto R;
return;
R:
if not t then
  | q;
  | goto L;
```

Validation

Working on `coreutils`

We chose to work on `mp_factor_using_pollard_rho`:

- 91 lines of code (before macro expansion)
- loops nested three levels deep
- has a `break` statement inside
- also contains a `goto` for error handling
- no indicator variables (yet)

Validation

Working on `coreutils`

First experiment:

- Blind, compile (clang), and then decompile (Ghidra).
- Decompiled code has 3 more `goto`'s and labels.
- Some manual effort to remove decompilation artifacts.
- Compare original code to decompiled code: OK.

Validation

Working on `coreutils`

Second experiment:

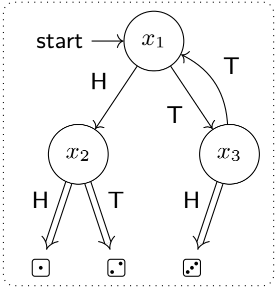
- Blind, then eliminate `goto` using indicators (Erosa & Hendren 1994, Cassé et al. 2002)
- Some manual effort to make indicator detection work.
- Compare original code to refactored code: OK.

See (Zhang, K., Narváez & Naus '25) for more.

Knuth-Yao algorithm

How to simulate  using  ?

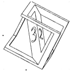
```
while true do
    if flip(0.5) then
        if flip(0.5) then
            return 1 // heads-heads
        else
            return 2 // heads-tails
    else
        if flip(0.5) then
            return 3 // tails-heads
        else
            skip // tails-tails
end
```



Correctness of Knuth-Yao in ProbGKAT

```
while true do
   if flip(0.5) then
      if flip(0.5) then
         return 1 // heads-heads
      else
         return 2 // heads-tails
   else
      if flip(0.5) then
         return 3 // tails-heads
      else
         skip // tails-tails
end
```

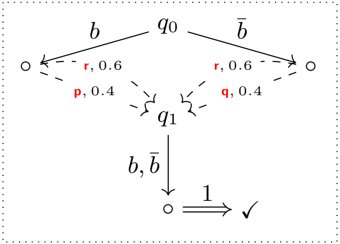
$\stackrel{?}{\equiv}$



```
if flip(1/3) then
   return 1
else
   if flip(0.5) then
      return 2
   else
      return 3
   end
end
```

See (Różowski, K., Kozen, Schmid & Silva '23) for more.

Operational model



- Notion of equivalence: coalgebraic bisimulation
- Can be decided in $O(n^2 \log(n))$ using a generic minimization algorithm (Wißmann et al. '20)

Thanks to ...

Balder ten Cate

Nate Foster

Justin Hsu

Dexter Kozen

David Narváez

Nico Naus

Wojciech Różowski

Todd Schmid

Alexandra Silva

Steffen Smolka

Cheng Zhang