On propositonal program equivalence

Tobias Kappé

SEN Symposium, 2025

Universiteit Leiden
The Netherlands

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
```

≡

```
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
```

Guarded Kleene Algebra with Tests (GKAT)

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

$$\mathrm{Exp} \ni e, f ::= \textbf{assert } b \mid p \in \Sigma \mid e; f \mid \textbf{if } b \textbf{ then } e \textbf{ else } f \mid \textbf{while } b \textbf{ do } e$$

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

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

GKAT semantics

Guarded strings

Inherited from KAT: sets of *guarded strings*, i.e., words of the form

$$\alpha_1 p_1 \alpha_2 p_2 \alpha_3 p_3 \cdots \alpha_n p_n \alpha_{n+1}$$

where the *atom* $\alpha_i \in \mathsf{At}$ tells us the truth value of each test at point i.

GKAT semantics

By example

$$\left[\!\left[\begin{array}{l}\texttt{if } b \texttt{ then}\\ \quad\mid\quad p;\\ \texttt{else}\\ \quad\mid\quad q;\\ \texttt{end}\end{array}\right]\!\right] = \left\{\begin{array}{l} bpb,\\ bp\overline{b},\\ \overline{b}q\overline{b},\\ \overline{b}qb \end{array}\right\}$$

GKAT semantics

By example

$$
\left\llbracket
\begin{array}{l}
\texttt{while } b \textbf{ or } c \texttt{ do} \\
\quad \texttt{if } b \texttt{ then} \\
\qquad p; \\
\quad \texttt{else} \\
\qquad q; \\
\quad \texttt{end} \\
\texttt{end}
\end{array}
\right\rrbracket
=
\left\{
\begin{array}{l}
\bar{b}\bar{c}, \\
\bar{b}cq\bar{b}\bar{c}, \\
b\bar{c}p\bar{b}cq\bar{b}\bar{c}, \\
\dots
\end{array}
\right\}
$$

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

$\mapsto$

$$q_0 \Longrightarrow \overline{\mathsf{b}}\,\overline{\mathsf{c}}$$

$$\begin{array}{l} \mathsf{bc} \mid \mathsf{p} \\ \mathsf{b}\overline{\mathsf{c}} \mid \mathsf{p} \\ \overline{\mathsf{b}}\mathsf{c} \mid \mathsf{q} \end{array}$$

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

GKAT decision procedure

GKAT automata equivalence

Theorem (Smolka et al. '20)

For every GKAT expression e, we can construct a GKAT automaton A_e such that $[\![e]\!] = [\![A_e]\!]$.

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.

Goto removal

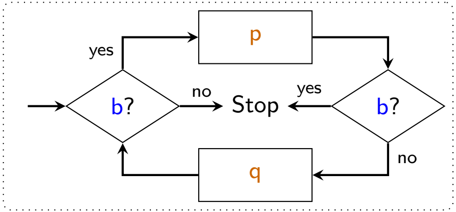
```
L:
if b then
  p;
  goto R;
return;
R:
if not b then
  q;
  goto L;
return;
```

goto removal
(e.g., calipso) →

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

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

Decompilation



GHIDRA

```
while b do
    p;
    if b 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:

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

CF-GKAT semantics

By example

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

$$
\left\llbracket \text{the program above} \right\rrbracket = \left\{ \begin{array}{l} bp\bar{b}q\bar{b}, \\ bp\bar{b}qbp\bar{b}q\bar{b}, \\ bpb \\ bp\bar{b}qbpb \\ \dots \end{array} \right\}
$$

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

$$
\left\llbracket \text{the program above} \right\rrbracket = \left\{ \begin{array}{l} bpb \\ bp\bar{b}q\bar{b}, \\ bp\bar{b}qbp\bar{b}q\bar{b}, \\ bp\bar{b}qbpb \\ \dots \end{array} \right\}
$$

Decision procedure

CF-GKAT automata

if b **and** $x = 1$ **then**
  $x := 2$;
  L:
  p;
else
  $x := 1$;
  goto L;

$\mapsto$

$$\begin{tikzcd}
(q_0, 2) \arrow[dr, bend left, "1 / p"] & \\
(q_0, 1) \arrow[r, "\bar{b} \mid p"] \arrow[d, "b \mid p"] & (q_1, 1) \arrow[d, Rightarrow] \\
(q_1, 2) \arrow[r, Rightarrow] & 1
\end{tikzcd}$$

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

For every CF-GKAT expression e, we can construct a GKAT automaton A_e such that $[\![e]\!] = [\![A_e]\!]$.

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

Lemma

A_e is linear in the size of e and $|I|$.

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 b then
  p;
  goto R;
return;
R:
if not b 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.

Expressivity

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

Theorem (Schmid, K., Kozen & Silva '21)

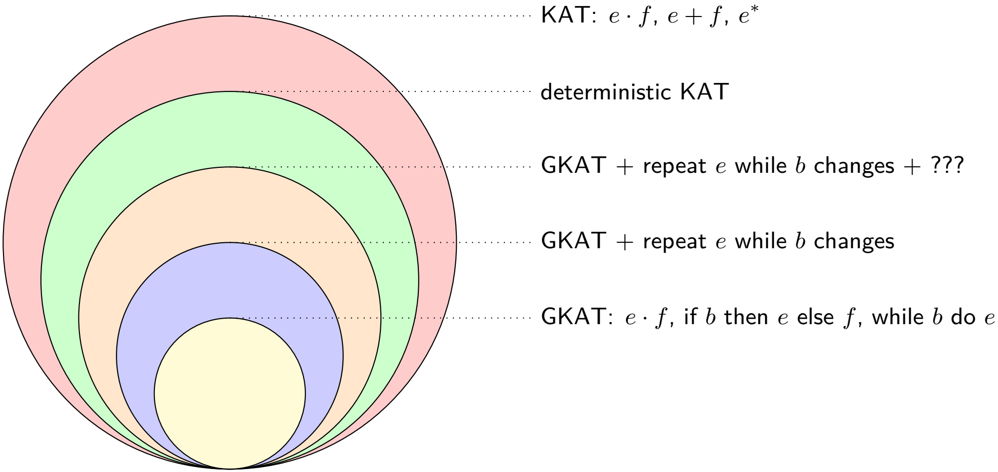
There is no program e built using

- `if-then-else`;
- `while-do`; *and*
- *sequential composition*

equivalent to the program on the left.

See also (Knuth & Floyd '71; Ashcroft & Manna '72; Peterson et al. '73; Kosaraju '74; Kozen & Tseng '08).

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

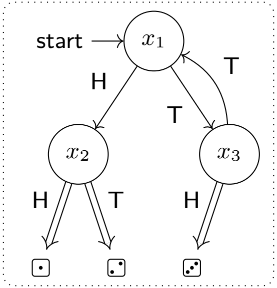
Theorem (Ten Cate & K. '25)

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

Knuth-Yao algorithm

How to simulate  using   ?

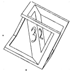
```
while true do
    if flip(0.5) then
        if flip(0.5) then
            return 1 // heads-heads
        else
            return 2 // heads-tails
    else
        if flip(0.5) then
            return 3 // tails-heads
        else
            skip // tails-tails
end
```



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
```

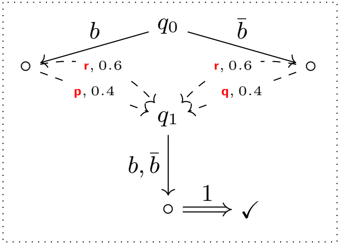
$\overset{?}{\equiv}$



```
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, Dorsch, Milius & Schröder '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