CF-GKAT

Efficient Validation of Control-Flow Transformations

Tobias Kappé

NetTCS @ LIACS, October 16, 2024

Universiteit Leiden
The Netherlands

Funded by
the European Union

Leiden Institute of Advanced Computer Science

Joint work with

Cheng Zhang
University College London

David E. Narváez
Virginia Tech

Nico Naus
OU & Virginia Tech

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
```

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)

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

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

- We can embed simple propositional programs:

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

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

- Language semantics in terms of *guarded strings*.
- Complete and finitary axiomatization
- Non-determinism makes equivalence PSPACE-complete
  - Basically: translate to automaton and check bisimilarity

(Kozen 1996), (Kozen & Smith 1996)

Guarded Kleene Algebra with Tests (GKAT)

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*.
- Complete but infinitary axiomatization (open problem)
- Language equivalence is efficiently decidable! (nearly linear)
  - Same idea as in KAT, but different composition operators.

(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* $\alpha_i \in At$ tells us the truth value of each test at point i.

GKAT semantics

By example

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

GKAT semantics

By example

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

GKAT decision procedure

GKAT automata

This comes from (Smolka et al. 2020).

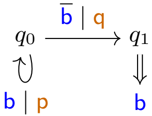
Let Q be a set. The set of *GKAT dynamics* on Q, denoted Q, consists of functions

$$At \to \bot + \checkmark + \Sigma \times Q$$

A *GKAT automaton* is a pointed G-coalgebra.

That is, it is a tuple (Q, δ, q_0) where:

- Q is a (finite) set of states;
- $q_0 \in Q$ is the *initial state*; and
- $\delta : Q \to GQ$ is the *transition function*.



Straightforward semantics in terms of guarded languages.

GKAT decision procedure

Translating to GKAT automata

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

$\mapsto$

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

$\circlearrowleft$

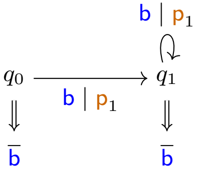
$bc \mid p$

$b\overline{c} \mid p$

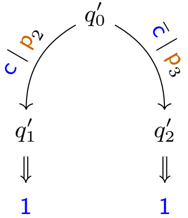
$\overline{b}c \mid q$

GKAT decision procedure

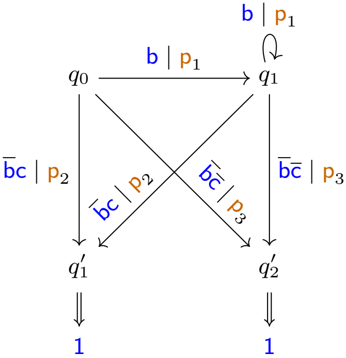
Translating to GKAT automata



$e = $ **while** b **do** p_1



$f = $ **if** c **then** p_2 **else** p_3



$g = e \cdot f$

Theorem

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.

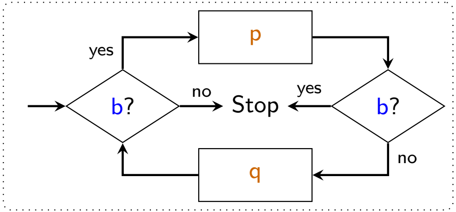
Goto removal

```
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
```

Decompilation



GHIDRA

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

Map of the Problematique

- GKAT does not support labels, **goto**, **return**, **break**, variables...
- It cannot even express the example programs! (Schmid, K. & Silva 2021)
- They *can* be translated to KAT expressions (Kozen 2008), e.g.:

$$(\mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q})^*(\mathsf{b}\mathsf{p}\mathsf{b} + \overline{\mathsf{b}})$$

- This is a rather non-trivial process...
- ...and KAT equivalence is PSPACE-hard...
- ...and really, we just need an automaton.

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:

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

CF-GKAT semantics

Guarded strings with continuations

First step: *guarded strings with continuations*, i.e.,

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

where c is a *continuation* of the form:

$$\mathbf{acc}\ i \qquad \mathbf{brk}\ i \qquad \mathbf{ret} \qquad \mathbf{jmp}\ (\ell, i)$$

A continuation tells us how a partial computation can proceed.

CF-GKAT semantics

By example

$$\left[\!\!\left[\begin{array}{l}\mathsf{p};\\ \textbf{if } \mathsf{b} \textbf{ then}\\ \quad \textbf{break};\\ \mathsf{q};\end{array}\right]\!\!\right]^{\sharp}_{i} = \left\{\begin{array}{l}\mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\overline{\mathsf{b}} \cdot \mathbf{acc}\ i,\\ \mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\mathsf{b} \cdot \mathbf{acc}\ i,\\ \mathsf{b}\mathsf{p}\mathsf{b} \cdot \mathbf{brk}\ i\end{array}\right\}$$

$$\left[\!\!\left[\begin{array}{l}\textbf{while } \mathsf{b} \textbf{ do}\\ \quad \mathsf{p};\\ \quad \textbf{if } \mathsf{b} \textbf{ then}\\ \quad\quad \textbf{break};\\ \quad \mathsf{q};\\ \textbf{end}\end{array}\right]\!\!\right]^{\sharp}_{i} = \left\{\begin{array}{l}\mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\overline{\mathsf{b}} \cdot \mathbf{acc}\ i,\\ \mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\overline{\mathsf{b}} \cdot \mathbf{acc}\ i,\\ \mathsf{b}\mathsf{p}\mathsf{b} \cdot \mathbf{acc}\ i\\ \mathsf{b}\mathsf{p}\overline{\mathsf{b}}\mathsf{q}\mathsf{b}\mathsf{p}\mathsf{b} \cdot \mathbf{acc}\ i\\ \dots\end{array}\right\}$$

CF-GKAT semantics

By example

$$\left[\!\left[\begin{array}{l} \texttt{L}: \\ \textbf{if } \mathsf{b} \textbf{ then} \\ \quad \mathsf{p}; \\ \quad \textbf{goto } \texttt{R}; \\ \textbf{return}; \\ \texttt{R}: \\ \textbf{if not } \mathsf{b} \textbf{ then} \\ \quad \mathsf{q}; \\ \quad \textbf{goto } \texttt{L}; \\ \textbf{return}; \end{array}\right]\!\right]^{\sharp}_{i} = \left\{ \begin{array}{l} \mathsf{bpb} \cdot \textbf{jmp}\ (\texttt{R}, i), \\ \mathsf{bp}\overline{\mathsf{b}} \cdot \textbf{jmp}\ (\texttt{R}, i), \\ \overline{\mathsf{b}} \cdot \textbf{ret} \end{array} \right\}$$

$$\left[\!\left[\begin{array}{l} \texttt{L}: \\ \textbf{if } \mathsf{b} \textbf{ then} \\ \quad \mathsf{p}; \\ \quad \textbf{goto } \texttt{R}; \\ \textbf{return}; \\ \texttt{R}: \\ \textbf{if not } \mathsf{b} \textbf{ then} \\ \quad \mathsf{q}; \\ \quad \textbf{goto } \texttt{L}; \\ \textbf{return}; \end{array}\right]\!\right]^{\texttt{R}}_{i} = \left\{ \begin{array}{l} \overline{\mathsf{b}}\mathsf{qb} \cdot \textbf{jmp}\ (\texttt{L}, i), \\ \overline{\mathsf{b}}\mathsf{q}\overline{\mathsf{b}} \cdot \textbf{jmp}\ (\texttt{L}, i), \\ \mathsf{b} \cdot \textbf{ret} \end{array} \right\}$$

CF-GKAT semantics

Resolution

Now we have a semantics of the form

$$[\![-]\!] : L \to I \to \text{guarded languages with continuations}$$

where guarded words end at the first **goto** statement.

What we need is a semantics of the form

$$[\![-]\!]{\downarrow} : L \to I \to \text{guarded languages}$$

where the jumps are "resolved" by stringing them together.

CF-GKAT semantics

Resolution

Resolution, in symbols:

$$\frac{w \cdot c \in [\![P]\!]_i^\ell \qquad c \in \{\mathbf{acc}\ j, \mathbf{ret}\}}{w \in [\![P]\!]{\downarrow}_i^\ell} \qquad \frac{w\alpha \cdot \mathbf{jmp}\ (\ell', j) \in [\![P]\!]_i^\ell \qquad \alpha x \in [\![P]\!]{\downarrow}_j^{\ell'}}{w\alpha x \in [\![P]\!]{\downarrow}_i^\ell}$$

$P =$

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

Then we can infer:

$$\frac{\mathsf{b} \cdot \mathbf{ret} \in [\![P]\!]_i^{\mathtt{R}}}{\mathsf{b} \in [\![P]\!]{\downarrow}_i^{\mathtt{R}}}$$

$$\frac{\mathsf{bpb} \cdot \mathbf{jmp}\ (\mathtt{R}, i) \in [\![P]\!]_i^\sharp \qquad \mathsf{b} \in [\![P]\!]{\downarrow}_i^{\mathtt{R}}}{\mathsf{bpb} \in [\![P]\!]{\downarrow}_i^\sharp}$$

Let C be the set of continuations (e.g., $\mathbf{acc}\ i$, $\mathbf{brk}\ i$, etc).

A *CF-GKAT dynamics* on a set X is a function of the form

$$I \to At \to \bot + C + \Sigma \times X \times I$$

We write GX for the set of CF-GKAT dynamics on X.

A CF-GKAT automaton is a tuple $A = (Q, \delta, q, \lambda)$ where

- Q is a set of *states* with $q \in Q$ the *initial state*
- $\delta : Q \to GQ$ is the *transition* function
- $\lambda : L \to GQ$ is the *jump* function

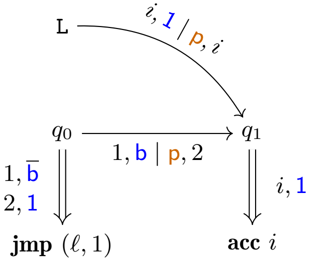
Has a straightforward semantics $[\![A]\!] : L \to I \to$ guarded languages with continuations.

Decision procedure

CF-GKAT automata

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

$\mapsto$



Decision procedure

Translating to CF-GKAT automata

Theorem

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

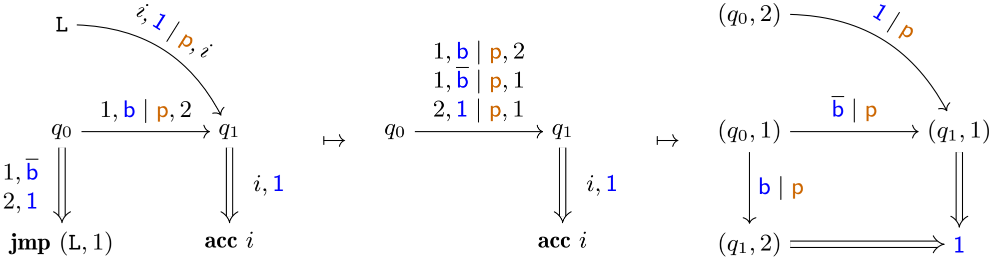
Another syntax-directed translation, à la (Thompson 1968).

Lemma

A_e is at most linear in the size of e.

Decision procedure

Translating to GKAT automata



Decision procedure

Translating to GKAT automata

Theorem

For every CF-GKAT automaton A, we can construct a GKAT automaton $A{\downarrow}$ such that $[\![A]\!]{\downarrow} = [\![A{\downarrow}]\!]$.

Lemma

$A{\downarrow}$ is at most $|I|$ times as large as A

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.

Conclusion & Further Work

- More experiments are necessary!
  - Automated testing pipeline
  - Scale to all of `coreutils`, Linux kernel, ...
  - Extract control flow from binary (no compilation).
- Expand support for top-level syntax:.
  - the **continue** statement
  - multi-way branching using **switch**
  - **do-while** loops with **break**
- Integration with existing decompiler
  - Baked-in translation validation
  - Continuous integration