Funded by
the European Union

Algebras for Deterministic Computation Are Inherently Incomplete

Balder ten Cate & Tobias Kappé

POPL, January 23rd 2025

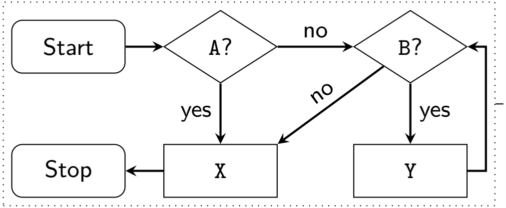
Universiteit Leiden
The Netherlands

liacs Leiden Institute of Advanced Computer Science

University of Amsterdam

Institute for Logic, Language and Computation

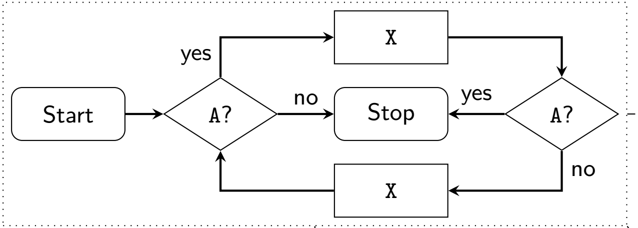
Flow control




```
if (!A()) {
  while (B()) {
    Y();
  }
}
X();
```

Expressivity

"if `A`, repeat `X` while `A` changes"




```
while (A()) {
  X();
  if (A()) break;
  X();
}
```

```
bool flag = true;
while (flag == A()) {
  X();
  flag = !flag;
}
```

```
start: if (!A()) return;
       X();
       if (A()) return;
       X();
       goto start;
```

We *need* non-local flow control for this program.

— see also (Knuth and Floyd 1971; Ashcroft and Manna 1972; Kozen and Tseng 2008; Schmid et al. 2021)

Expressivity

Just `if-then-else` and `while-do` are not enough; what do we to need to express everything?

- Single-level `break` helps, but is not enough (Kosaraju 1974; Kozen and Tseng 2008)
- Multi-level `break` lets us express everything (Kosaraju 1974; Kozen 2008)
- Having variables also suffices (Böhm and Jacopini 1966; Grathwohl et al. 2014)
- Obviously, having `goto` or tail recursion is also enough!

Each of these options has its own issues:

- `break` obscures loop conditions;
- multi-level `break` even more so (and is rare);
- `goto` can lead to "spaghetti code";
- using variables makes control flow implicit;
- non-trivial tail recursion may scatter your code.

Main result

Are there *local* control-flow primitives
that can express *all* deterministic control flow?

(e.g., maybe `if-then-else`, `while-do`, *and* `repeat-while-changes`?)

No! *

* unless you allow infinitely many of them

Formalization

We need a language to denote control flow:

$$\mathsf{BA} \ni b, c ::= \mathsf{false} \mid \mathsf{true} \mid t \in T \mid b \;\mathsf{or}\; c \mid b \;\mathsf{and}\; c \mid \mathsf{not}\; b$$
$$\mathsf{KAT} \ni e, f ::= b \in \mathsf{BA} \mid p \in \Sigma \mid e + f \mid e \cdot f \mid e^*$$

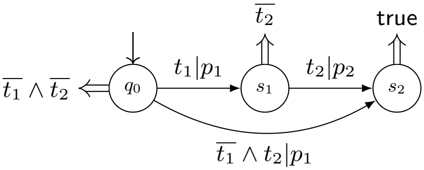
KAT can express traditional (deterministic) control flow:

$$\mathsf{if}\; b \;\mathsf{then}\; e \;\mathsf{else}\; f := b \cdot e + (\mathsf{not}\; b) \cdot f \qquad \mathsf{while}\; b \;\mathsf{do}\; e := (b \cdot e)^* \cdot (\mathsf{not}\; b)$$

A (parametrized) relational semantics:

$$\mathcal{R}[\![-]\!] : \mathsf{KAT} \to \forall S : \mathsf{Set}, (\underbrace{T \to 2^S}_{\text{test interp. } \tau}) \to (\underbrace{\Sigma \to 2^{S \times S}}_{\text{action interp. } \sigma}) \to 2^{S \times S}$$

Automata model



Automata like these are exactly as expressive as KAT (Kozen 2003).

Determinism

Recall that we were interested in *deterministic* flow control.

Theorem

Let $e \in \mathsf{KAT}$. The following are equivalent:

1. *if each $\sigma(p)$ is a partial function, then so is $\mathcal{R}[\![e]\!]^{\sigma}_{\tau}$;*
2. *the automaton for e is deterministic.*

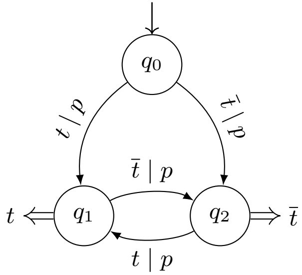
We have already seen some deterministic expressions:

$$\text{if } b \text{ then } e \text{ else } f := b \cdot e + (\text{not } b) \cdot f \qquad\qquad \text{while } b \text{ do } e := (b \cdot e)^* \cdot (\text{not } b)$$

The notion of "determinism" for KAT expressions is robust!

Custom flow control

Here is an automaton for "`repeat` p `while` t `changes`":



The corresponding (deterministic) KAT expression is

$$tp(\bar{t}ptp)^*(t + \bar{t}p\bar{t}) + \bar{t}p(tp\bar{t}p)^*(\bar{t} + tpt)$$

Custom flow control

We now have a new primitive for deterministic flow control:

$$\text{repeat } e \text{ while } b \text{ changes} := be(\bar{b}ebe)^*(b + \bar{b}e\bar{b}) + \bar{b}e(be\bar{b}e)^*(\bar{b} + beb)$$

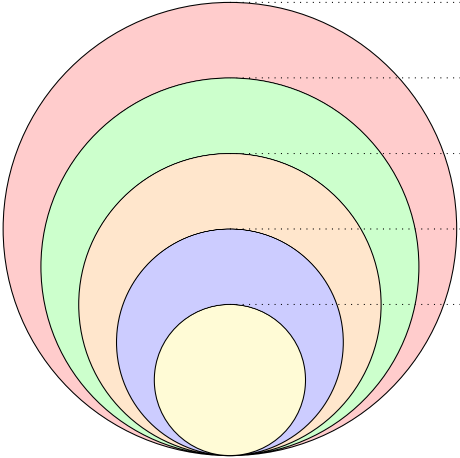
Theorem (cf. Kozen and Tseng 2008; Schmid et al. 2021)

There is no e built using `if-then-else` *and* `while-do` *and sequential composition such that*

$$\mathcal{R}[\![e]\!] = \mathcal{R}[\![\text{repeat } p \text{ while } t \text{ changes}]\!]$$

See also (Knuth and Floyd 1971; Ashcroft and Manna 1972; Peterson et al. 1973; Kosaraju 1974).

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

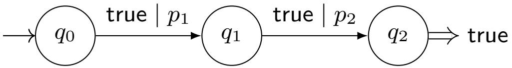
Main result

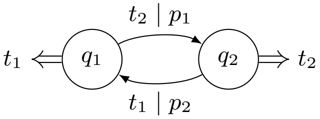
Theorem

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.*

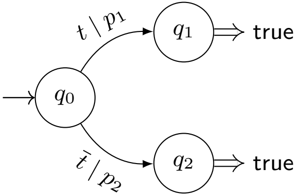
Proof idea

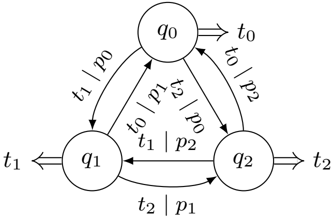
$p_1 \cdot p_2 \equiv$



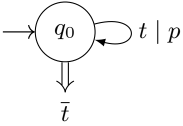


if t then p_1 else $p_2 \equiv$





while t do $p \equiv$



Further work

- The operators of GKAT are at most 1-dense. Is GKAT characterized by such automata?
- Is this result still true for extensions of KAT, like dKAT, or KAT with intersection?
- How does our hierarchy compare to Kosaraju's `break` hierarchy?
- Is there a different kind of composition that can incorporate, say, n-level breaks?