Concurrent NetKAT

Jana Wagemaker, Nate Foster, Tobias Kappé
Dexter Kozen, Jurriaan Rot, Alexandra Silva
Radboud Universiteit, Cornell University, ILLC, University of Amsterdam
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs

Many variants and extensions that enable analysis of more complicated programs:
  ▶ Kleene algebra with tests (KAT)
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs

Many variants and extensions that enable analysis of more complicated programs:
  ▶ Kleene algebra with tests (KAT)
  ▶ NetKAT
  ▶ …
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs

Many variants and extensions that enable analysis of more complicated programs:
  ▶ Kleene algebra with tests (KAT)
  ▶ NetKAT
  ▶ …

Key limitation of NetKAT: stateless and sequential
Sequential programs; see e.g. [Kleene 1956], [Salomaa 1966], and [Kozen 1994].

Programs w/ concurrency; see e.g. [Hoare et al. 2009], [Laurence and Struth 2014], and [K. et al. 2018].

A combination of both; see e.g. [Jipsen 2014], [Jipsen and Moshier 2016], and [O’Hearn et al. 2015].

Programs w/ flow control; see e.g. [Kozen 1996], [Kozen and Smith 1996], and [Kozen and Patron 2000].
\[ p \land q = p ; q \sim p \land q \leq p ; q \]

POCKA: fine-grained reasoning about concurrent programs with conditionals that manipulate a shared global memory.

\[
\begin{array}{c}
\text{KA} \\
\text{concurrency} \\
\end{array} \xrightarrow{\text{observations}} \begin{array}{c}
\text{KAO} \\
\end{array}
\]

\[
\begin{array}{c}
\text{CKA} \\
\end{array} \xrightarrow{\text{concurrency}} \begin{array}{c}
\text{CKAO} \\
\end{array}
\]
\[ p \land q = p ; q \Rightarrow p \land q \leq p ; q \]

\[ KA \xrightarrow{\text{observations}} KAO \]

\[ \text{concurrency} \]

\[ CKA \xrightarrow{\text{concurrency}} \text{CKAO} \]

⇒ POCKA: fine-grained reasoning about concurrent programs with conditionals that manipulate a shared global memory.
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs

Many variants and extensions that enable analysis of more complicated programs:
  ▶ Kleene algebra with tests (KAT)
  ▶ NetKAT
  ▶ Concurrent extensions (CKA, POCKA, CKAO, …)
  ▶ …

Key limitation of NetKAT: stateless and sequential
Kleene algebra: axiomatisation of regular languages, used to reason about simple programs

Many variants and extensions that enable analysis of more complicated programs:

▶ Kleene algebra with tests (KAT)
▶ NetKAT
▶ Concurrent extensions (CKA, POCKA, CKAO, …)
▶ …

Key limitation of NetKAT: stateless and sequential

This paper: Concurrent NetKAT, combination of POCKA and a multi-packet extension of NetKAT

▶ Sound and complete axiomatisation of CNetKAT
▶ Examples of applicability of CNetKAT for modelling and analysing concurrent network behaviours
Do we always see a green packet at switch /three.osf before we see a red packet at switch /two.osf?
Do we always see a green packet at switch 3 before we see a red packet at switch 2?
A packet is a record of fields $f_1, \ldots, f_N$ and values $v_1, \ldots, v_N$. 
A packet is a record of fields $f_1, \ldots, f_N$ and values $v_1, \ldots, v_N$. Part of syntax:

$$f = n \mid f \leftarrow n \mid v = n \mid v \leftarrow n \mid v \leftarrow v'$$
A packet is a record of fields $f_1, \ldots, f_N$ and values $v_1, \ldots, v_N$. Part of syntax:

$$f = n \mid f \gets n \mid v = n \mid v \gets n \mid v \gets v'$$

More complicated programs with $+ \quad ; \quad * \quad ||$
/one.osf := /two.osf; (tag := red; sw ← /two.osf) ∥ (tag := green; sw ← /three.osf)) /seven.osf
\[ p_1 := sw=1; ((\text{tag} = \text{red} ; sw \leftarrow 2) \parallel (\text{tag} = \text{green} ; sw \leftarrow 3)) \]
sw = 1; ((v = 1; tag = red; sw ← 2) || (tag = green; sw ← 3; v ← 1))
\[ p \triangleq v \leftarrow 0 ; (p_1 \parallel p_2 \parallel p_3 \parallel p_4)^* \]
Type of Semantics

$J^p_k \rightarrow J^p_k \cdot \Pi_k^t \cdot u \cdot b \in J^p_k(a)$ means "there is an execution of $p$ on input set $a$ that changes the global variables according to $u$, and the set of output packets produced is $b"."

$J^p_k \parallel J^q_k(a) \equiv \{ (u \parallel v) \cdot (b \cup c) | u \cdot b \in J^p_k(a), v \cdot c \in J^q_k(a) \}$

$J^{dup}_k(a) \equiv \{ a \cdot a \}$

Full semantics: $J^p_k \downarrow (a)$
Type of Semantics

\[ [-] : 2^P_k \rightarrow 2^{Pom} \cdot 2^P_k \]

Full semantics:

\[ J_{p,K}(a) \]

\[ J_{dup}(a) \equiv \{ a \cdot a \} \]

\[ J_{p,K} \mid \downarrow (a) \]
Type of Semantics

\[ \llbracket \cdot \rrbracket : 2^P_k \rightarrow 2^{Pom} \cdot 2^P_k \]

\( u \cdot b \in \llbracket p \rrbracket (a) \) means “there is an execution of \( p \) on input set \( a \) that changes the global variables according to \( u \), and the set of output packets produced is \( b \)”
Type of Semantics

\[ \llbracket - \rrbracket : 2^\mathbb{P}_k \rightarrow 2^{\mathbb{P}_o \cdot 2^\mathbb{P}_k} \]

\[ u \cdot b \in \llbracket p \rrbracket (a) \text{ means “there is an execution of } p \text{ on input set } a \text{ that changes the global variables according to } u, \text{ and the set of output packets produced is } b” \]

\[ \llbracket p \parallel q \rrbracket (a) \triangleq \{ (u \parallel v) \cdot (b \cup c) \mid u \cdot b \in \llbracket p \rrbracket (a), v \cdot c \in \llbracket q \rrbracket (a) \} \]
Type of Semantics

\[ [-] : 2^{P_k} \to 2^{P_{om} \cdot 2^{P_k}} \]

\( u \cdot b \in \llbracket p \rrbracket(a) \) means “there is an execution of \( p \) on input set \( a \) that changes the global variables according to \( u \), and the set of output packets produced is \( b \)”

\[ \llbracket p \parallel q \rrbracket(a) \triangleq \{(u \parallel v) \cdot (b \cup c) \mid u \cdot b \in \llbracket p \rrbracket(a), v \cdot c \in \llbracket q \rrbracket(a)\} \]

\[ \llbracket \text{dup} \rrbracket(a) \triangleq \{a \cdot a\} \]
### Type of Semantics

\[ [-] : 2^{P_k} \rightarrow 2^{\text{Pom} \cdot 2^{P_k}} \]

\( u \cdot b \in \llbracket p \rrbracket(a) \) means “there is an execution of \( p \) on input set \( a \) that changes the global variables according to \( u \), and the set of output packets produced is \( b \)”

\[ \llbracket p \parallel q \rrbracket(a) \triangleq \{(u \parallel v) \cdot (b \cup c) \mid u \cdot b \in \llbracket p \rrbracket(a), v \cdot c \in \llbracket q \rrbracket(a)\} \]

\[ \llbracket \text{dup} \rrbracket(a) \triangleq \{a \cdot a\} \]

Full semantics: \( \llbracket p \rrbracket \downarrow(a) \)
CNetKAT contains the following axioms:
CNetKAT contains the following axioms:

- Kleene algebra axioms
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
- NetKAT packet axioms
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
- NetKAT packet axioms
- A Boolean algebra for packet tests
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
- NetKAT packet axioms
- A Boolean algebra for packet tests
- A PCDL for the observations
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
- NetKAT packet axioms
- A Boolean algebra for packet tests
- A PCDL for the observations
- Axioms connecting the BA and PCDL operators to the Kleene algebra ones
CNetKAT contains the following axioms:

- Kleene algebra axioms
- Axioms for the parallel from concurrent Kleene algebra
- NetKAT packet axioms
- A Boolean algebra for packet tests
- A PCDL for the observations
- Axioms connecting the BA and PCDL operators to the Kleene algebra ones
- Axioms connecting the local and the global state
drop $\parallel p = p$ \hspace{2cm} \text{abort } \parallel p = \text{abort}
Axiomatisation

\begin{align*}
\text{drop} \parallel p &= p \\
\text{abort} \parallel p &= \text{abort} \\

\text{if} \; t \lor t' = t \parallel t' \\
\text{then} \; o \lor o' &= o + o' 
\end{align*}
\( o \leq o' \iff o \land o' = \bot \)
\[ o \leq o' \iff o \land o' = \bot \]

\([v = n]\) contains all partial functions that assign \(n\) to \(v\).
\[ o \leq o' \iff o \land o' = \perp \]

\([v = n]\) contains all partial functions that assign \(n\) to \(v\).

\([\overline{v = n}]\) should contain all partial functions that assign a value to \(v\) different than \(n\).
Theorem

Let \( p, q \in \text{Prg} \). We have \( \text{CNetKAT} \vdash p = q \) if and only if \( \llbracket p \rrbracket_\downarrow = \llbracket q \rrbracket_\downarrow \).
1. Define a normal form for CNetKAT programs:

\[ \Pi_a \cdot p = \Pi_a \cdot \sum_{j \in J} (u_j \cdot \Pi_{b_j}) \]
1. Define a normal form for CNetKAT programs:

\[ \Pi_a ; p = \Pi_a ; \sum_{j \in J} (u_j ; \Pi_{b_j}) \]

2. Obtain completeness for \( \Pi_a \)-shaped programs from NetKAT completeness.
4. **Four parts of completeness proof**

1. Define a normal form for CNetKAT programs:

   \[ \Pi_a ; p = \Pi_a ; \sum_{j \in J} (u_j ; \Pi_{b_j}) \]

2. Obtain completeness for \( \Pi_a \)-shaped programs from NetKAT completeness.

3. Using completeness of POCKA, obtain completeness for programs of the form \( s ; \Pi_a \) (and sums thereof), where \( s \) is a state program.
1. Define a normal form for CNetKAT programs:

\[ \Pi_a p = \Pi_a \sum_{j \in J} (u_j ; \Pi_b) \]

2. Obtain completeness for \( \Pi_a \)-shaped programs from NetKAT completeness.

3. Using completeness of POCKA, obtain completeness for programs of the form \( s ; \Pi_a \) (and sums thereof), where \( s \) is a state program.

4. Lastly, we combine these results to prove that if \( p \) and \( q \) have the same behaviour on input \( a \), the program \( \Pi_a p \) is provably equivalent to \( \Pi_a q \). We can conclude using the extensionality axiom: \( \forall a \in 2^{pk}.(\Pi_a p = \Pi_a q) \Rightarrow p = q \).
How to study isolated behaviour? –/greater.osf guarded pomsets
$v \leftarrow 1; v = 2$
How to study isolated behaviour? –/greater.osf guarded pomsets

\[ v \leftarrow 1; v = 2 \]

\[(v \leftarrow 1 ; v = 2) \parallel v \leftarrow 2\]
How to study isolated behaviour? \( \rightarrow \) **guarded pomsets**

\[
v \leftarrow 1; v = 2 \\
(v \leftarrow 1; v = 2) \parallel v \leftarrow 2
\]
With these tools we can analyse example from beginning
Final Remarks

- With these tools we can analyse example from beginning
- Missing: a decision procedure
With these tools we can analyse example from beginning

- Missing: a decision procedure

- Future work: lots of more case studies