Equational Theories of Abnormal Termination Based on Kleene Algebra

Konstantinos Mamouras(✉)

University of Pennsylvania, Philadelphia, USA
mamouras@seas.upenn.edu

Abstract. We study at an abstract level imperative while programs with an explicit *fail* operation that causes abnormal termination or irreparable failure, and a *try-catch* operation for error handling. There are two meaningful ways to define the semantics of such programs, depending on whether the final state of the computation can be observed upon failure or not. These two semantics give rise to different equational theories. We investigate these two theories in the abstract framework of Kleene algebra, and we propose two simple and intuitive equational axiomatizations. We prove very general conservativity results, from which we also obtain decidability and deductive completeness of each of our calculi with respect to the intended semantics.

1 Introduction

The computations of imperative programs are typically divided into two distinct categories: those that terminate normally at some final state (thus possibly yielding an output), and those that do not terminate or, as we say, that diverge. However, for most realistic programs there is also the possibility of *failure*, which has to be distinguished from normal termination. When we say failure here, we are referring to the computational phenomenon where an executing program has to stop immediately because something "bad" has happened that prevents it from continuing with its computation. There are numerous examples of such behavior: a memory access error, a division by zero, the failure of a user-defined assertion, and so on. Depending on the context, this kind of irreparable failure is described with various names: abnormal or abrupt termination, (uncaught) exception, program crash, etc.

An important point to be made is that when failure is a possibility there are two different (both very meaningful) ways of defining semantics, depending on whether the final state of the computation can be observed upon failure or not. Let us consider a standard way of describing the intended input-output behavior of imperative programs by describing how they execute on an idealized machine. This is the so called *operational semantics*, and it amounts to giving a detailed description of the individual steps of the computation as it mutates the program state. In our setting, where failure is a possibility, a computation of the program f can take one of the following three forms:

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 88–105, 2017.
DOI: 10.1007/978-3-662-54458-7_6

$$\begin{aligned}\text{normal termination: } & \langle x, f\rangle \to \langle x', f'\rangle \to \cdots \to \langle y, 1\rangle \\ \text{divergence: } & \langle x, f\rangle \to \langle x', f'\rangle \to \cdots \\ \text{abnormal termination: } & \langle x, f\rangle \to \langle x', f'\rangle \to \cdots \to \langle y, \mathsf{fail}(e)\rangle\end{aligned}$$

The letters x, x', y above represent entire program states, the constant 1 represents the program that immediately terminates normally, and the constant $\mathsf{fail}(e)$ represents the program that immediately terminates abnormally with error code (exception) e. We summarize the input-output behavior of the program f by including the pair $x \mapsto \langle y, 1\rangle$ when the normally terminating computation $\langle x, f\rangle \to \cdots \to \langle y, 1\rangle$ is possible, and the pair $x \mapsto \langle y, \mathsf{fail}(e)\rangle$ when the abnormally terminating computation $\langle x, f\rangle \to \cdots \to \langle y, \mathsf{fail}(e)\rangle$ is possible. Thus, the *meaning* or *denotation* of a program f that computes by transforming a state space X is given by a function of type $X \to P(X \oplus (X \times E))$, where P denotes the powerset functor, E is the set of error codes (exceptions), and $\oplus$ is the coproduct (disjoint union) operation. Intuitively, using this semantics we are able to observe the final state of the computation upon abnormal termination. However, it may sometimes be appropriate to disregard the final state when the program fails. In this latter case, the meaning of the program is given by a function of type $X \to P(X \oplus E)$. Informally, our second semantics is derived from the first one by "forgetting" the final state in the case of failure. To sum this up:

$$\begin{aligned}\text{Final state } \textit{observable} \text{ upon failure: } \quad & X \to P(X \oplus (X \times E)) \\ \text{Final state } \textit{not observable} \text{ upon failure: } \quad & X \to P(X \oplus E)\end{aligned}$$

In both cases, the powerset functor P allows us to accomodate nondeterminism. That is, when a program starts at some state, there may be several possible computations and we want to record all their possible outcomes.

These two different ways of assigning meaning to programs that we described in the previous paragraph give rise to ***two distinct notions of equivalence***. Let us write $f \equiv g$ if the programs f and g have the same meaning under the first semantics involving functions $X \to P(X \oplus (X \times E))$. We write $\approxeq$ to denote the equivalence induced by the semantics involving functions $X \to P(X \oplus E)$. An immediate observation is that the equivalence $\equiv$ is *finer* than $\approxeq$, which means that $f \equiv g$ implies $f \approxeq g$. In fact, $\equiv$ is *strictly finer* than $\approxeq$, because:

$$x := 0; \mathsf{fail}(e) \not\equiv x := 1; \mathsf{fail}(e) \qquad\qquad x := 0; \mathsf{fail}(e) \approxeq x := 1; \mathsf{fail}(e)$$

where ; is the sequential composition operation. The programs $x := 0; \mathsf{fail}(e)$ and $x := 1; \mathsf{fail}(e)$ both terminate abnormally at states with $x = 0$ and $x = 1$ respectively. When the final states can be observed, the two programs can be differentiated by looking at the final value of the variable x. But, when the final states are unobservable, then the programs are semantically the same.

We are interested here in axiomatizing completely the two equational theories given by the equivalences $\equiv$ and $\approxeq$. Of course, this endeavor is not possible for typical interpreted programming languages that are Turing-complete. For such languages the equational theories are not recursively enumerable, and hence they admit no complete effective axiomatization. We will therefore work in a very abstract uninterpreted setting, where the building blocks of our imperative programming language are abstract actions $a, b, c, \ldots$ with no fixed interpretation. This is the setting of Kleene algebra (KA) [12] and Kleene algebra with tests (KAT) [13], whose language includes the following program structuring operations: sequential composition $\cdot$, nondeterministic choice $+$, and nondeterministic iteration *. KAT involves a special syntactic category of *tests* $p, q, r, \ldots$, which can model the guards of conditionals and while loops:

$$\mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, q \triangleq p \cdot f + \neg p \cdot g \qquad\qquad \mathsf{while}\, p\, \mathsf{do}\, f \triangleq (p \cdot f)^* \cdot \neg p$$

It is therefore sufficient to consider the language of KAT for logical investigations that concern imperative nondeterministic programs with while loops.

If failure is not a possibility, then the axioms of KA and KAT are very well suited for abstract reasoning about programs. However, if failure is possible then the property $f \cdot 0 = 0$, which is an axiom of KA, is ***no longer valid***. The constant 0 denotes the program that always diverges, and we expect:

$$\mathsf{fail}(e) \cdot 0 = \mathsf{fail}(e) \qquad\qquad \mathsf{fail}(e) \cdot 0 \neq 0$$

This suggests that, in order to reason conveniently about programs that can fail, we need to enrich the syntax of KAT and capture the essential properties of the fail operation. As a first step, we introduce the simplest type discipline necessary by distinguishing the programs that contain no occurrence of fail. In algebraic jargon, this is a *two-sorted* approach: we consider a sub-sort of fail-*free* programs in addition to the sort of all programs. The basic algebraic theory that we propose, which we call **FailKAT**, differs from KAT by weakening the axiom $f \cdot 0 = 0$ and by introducing just one extra equation for $\mathsf{fail}(e)$:

$$\begin{aligned} f \cdot 0 &= 0, \text{ where } f \text{ is a } \mathsf{fail}\text{-free program} \\ \mathsf{fail}(e) \cdot f &= \mathsf{fail}(e), \text{ where } f \text{ can be any program} \end{aligned}$$

As we shall see, these modifications are exactly what is needed to reason about failure (for the $\equiv$ equivalence). They are absolutely necessary, and they introduce no additional syntactic complications than what is strictly required. Even more interestingly, we can axiomatize the coarser $\approx$ equivalence by adding to the **FailKAT** calculus just one more equational axiom (The partial order $\leq$ is defined in every KA in terms of nondeterministic choice as follows: $x \leq y$ iff $x + y = y$. See, for example, [12].):

$$f \cdot \mathsf{fail}(e) \leq \mathsf{fail}(e), \text{ where } f \text{ is a } \mathsf{fail}\text{-free program.}$$

Let us call this extended calculus **FailTKAT**. The above property has a straightforward intuitive explanation. When we are disallowed to observe the final state upon failure, then any program that we execute before failing has no observable effect other than possibly causing divergence (hence we have $\leq$ instead of $=$).

```
$i := 0$                                              $\equiv$    $i := 0$
while $(i < n)$ do                                                while $(i < n)$ do
        assert $(0 \leq i) \wedge (i < X.\mathsf{length})$                if $(i \geq X.\mathsf{length})$ then fail
        $X[i] := 0$                                                       $X[i] := 0$
        $i := i + 1$                                                      $i := i + 1$
```

$$(i := 0) \equiv (i := 0); (i \geq 0) \qquad (i \geq 0); (X[i] := 0) \equiv (X[i] := 0); (i \geq 0)$$

$$\neg(i < X.\mathsf{length}) \equiv (i \geq X.\mathsf{length}) \qquad (i \geq 0); (i := i + 1) \equiv (i \geq 0); (i := i + 1); (i \geq 0)$$

Fig. 1. The logical system **FailKAT** can be used for verifying program optimizations.

We further enrich the language with a *try-catch construct* for exception handling. The operational semantics that we consider is the standard one used in imperative programming languages:

$$\frac{\langle x, f\rangle \to \langle x', f'\rangle}{\langle x, \mathsf{try}\, f\, \mathsf{catch}(e)\, g\rangle \to \langle x', \mathsf{try}\, f'\, \mathsf{catch}(e)\, g\rangle} \qquad \begin{array}{l} \langle x, \mathsf{try}\, 1\, \mathsf{catch}(e)\, g\rangle \to \langle x, 1\rangle \\ \langle x, \mathsf{try}\, \mathsf{fail}(e)\, \mathsf{catch}(e)\, g\rangle \to \langle x, g\rangle \end{array}$$

$$\langle x, \mathsf{try}\, \mathsf{fail}(d)\, \mathsf{catch}(e)\, g\rangle \to \langle x, \mathsf{fail}(d)\rangle, \text{ when } d \neq e$$

As we will prove, the following five equational axioms

$$\begin{aligned} \mathsf{try}\, 1\, \mathsf{catch}(e)\, g &= 1 \\ \mathsf{try}\, \mathsf{fail}(d)\, \mathsf{catch}(e)\, g &= \mathsf{fail}(d), \text{ when } d \neq e \\ \mathsf{try}\, \mathsf{fail}(e)\, \mathsf{catch}(e)\, g &= g \\ \mathsf{try}\, (f + g)\, \mathsf{catch}(e)\, h &= (\mathsf{try}\, f\, \mathsf{catch}(e)\, h) + (\mathsf{try}\, g\, \mathsf{catch}(e)\, h) \\ \mathsf{try}\, (f \cdot g)\, \mathsf{catch}(e)\, h &= f \cdot (\mathsf{try}\, g\, \mathsf{catch}(e)\, h), \text{ when } f \text{ is fail-free} \end{aligned}$$

are sufficient to derive all the algebraic properties of the try-catch construct. We note that fail and try-catch can also be used to model several other useful non-local control flow constructs, such as break, continue and return.

The problem of axiomatizing the algebraic properties of the $\mathsf{try}\ \ldots\ \mathsf{catch}(e)\ \ldots$ and $\mathsf{fail}(e)$ constructs is more than of purely mathematical interest. There are several useful program optimizations that concern user-defined assertions and statements that can cause failure. **FailKAT** can offer a formal compositional approach to the verification of such program optimizations. For example, in Fig. 1 we see a valid program optimization that simplifies a user-defined assertion, where the statement $\mathsf{assert}\ p$ is syntactic sugar for $\mathsf{if}\, p\, \mathsf{then}\, 1\, \mathsf{else}\, \mathsf{fail}$. The optimization is expressed as an equation, and it can be formally proved correct using **FailKAT** and some additional equations (see bottom of Fig. 1) that encode the relevant properties of the domain of computation. Reasoning in KA and KAT under the presence of additional equational hypotheses is studied in [17,24]. Our results here are general enough to apply to this setting.

Our contribution. We study within the framework of Kleene algebra with tests [13] an explicit *fail* operation, which is meant to model the abnormal termination of imperative sequential programs, and a *try-catch* construct for handling exceptions. Our main results are the following:

– We show that every Kleene algebra with tests K can be extended conservatively with a family of $\mathsf{fail}(e)$ constants and $\mathsf{try} \ldots \mathsf{catch}(e) \ldots$ operations, where e ranges over the set E of possible exceptions. The resulting structure is the least restrictive extension of K that satisfies the axioms of **FailKAT**.
– From this conservativity theorem we obtain as a corollary a *completeness* theorem for the class of algebras of functions $X \to P(X \oplus (X \times E))$.
– The setting of **FailTKAT** requires an extended construction that works on a subclass of Kleene algebras with tests that possess a *top element* $\top$. We show how to extend conservatively such algebras with *fail* and *try-catch* operations, so that the extension is a **FailTKAT**.
– We then derive as a corollary the completeness of the calculus **FailTKAT** for the equational theory induced by the coarser semantics in terms of functions $X \to P(X \oplus E)$.

2 Relational Models of Failure

A standard denotational way of describing the meaning of (failure-free) sequential imperative programs is using binary relations or, equivalently, functions of type $X \to PX$, where X is the state space and P is the powerset functor [32]. This denotational semantics agrees with the intended operational semantics, and its advantage is that it allows us to define the meaning compositionally (by induction on the structure of the program) using certain algebraic operations on the carrier $X \to PX$. In the program semantics literature, this is usually referred to as the standard *relational semantics* of imperative programs.

In this section, we follow an analogous approach for the denotational semantics of programs that can terminate abnormally. As we have already discussed in the introduction, there are two different ways to define the meaning of such programs, which lead us to consider functions of type $X \to P(X \oplus (X \times E))$ and $X \to P(X \oplus E)$ respectively. We will endow these sets of functions with algebraic operations that are sufficient for interpreting the program structuring operations. Our treatment is infinitary, since we will be considering an arbitrary choice $\sum$ operation. This is more convenient for semantic investigations than a finitary treatment, because the language is more economical (the operations 0, $+$ and * can all be expressed in terms of $\sum$).

We write $\iota_1^{X,Y} : X \to X \oplus Y$ for the left injection map, and $\iota_2^{X,Y} : Y \to X \oplus Y$ for the right injection map. The superscript X, Y is omitted when the types can be inferred from the context. A function k of type $X \to P(X \oplus (X \times E))$ is said to be *fail-free* if $k(x)$ is contained in $\{\iota_1(x) \mid x \in X\}$ for all $x \in X$. For a set X, we define now the algebra $\mathcal{A}_X$ of all functions from X to $P(X \oplus (X \times E))$:

$$\begin{aligned}
\mathcal{A}_X &\triangleq (A, K, \cdot, \textstyle\sum, 1, (\mathit{fail}_e)_{e\in E}, (\mathit{try}\text{-}\mathit{catch}_e)_{e\in E}) & 1(x) &\triangleq \{\iota_1(x)\} \\
A &\triangleq \text{functions } X \to P(X \oplus (X \times E)) & \mathit{fail}_e(x) &\triangleq \{\iota_2(x, e)\} \\
K &\triangleq \{k \in A \mid k \text{ is fail-free}\} & (\textstyle\sum_i f_i)(x) &\triangleq \textstyle\bigcup_i f_i(x)
\end{aligned}$$

$$(f \cdot g)(x) \triangleq \bigcup\{g(y) \mid \iota_1(y) \in f(x)\} \cup \{\iota_2(y, e) \mid \iota_2(y, e) \in f(x)\}$$

$$\begin{aligned}
(\mathit{try}\, f\, \mathit{catch}_e\, g)(x) \triangleq{}& \{\iota_1(y) \mid \iota_1(y) \in f(x)\} \cup \bigcup\{g(y) \mid \iota_2(y, e) \in f(x)\} \cup \\
& \{\iota_2(y, d) \mid \iota_2(y, d) \in f(x) \text{ and } d \neq e\}
\end{aligned}$$

Definition 1 (F-Quantales). Fix a set E of exceptions. An *F-quantale* with exceptions E is a two-sorted algebraic structure

$$(A, K, \cdot, \textstyle\sum, 1, (\mathit{fail}_e)_{e\in E}, (\mathit{try}\text{-}\mathit{catch}_e)_{e\in E})$$

with carriers $K \subseteq A$, where K is the sort of *fail-free elements*, and A is the sort of all elements. We require that K is closed under $\cdot$ and $\sum$, and that the following hold (the variables $u, v, \ldots$ range over A and $x, y, \ldots$ range over K):

$$\begin{aligned}
(u \cdot v) \cdot w = u \cdot (v \cdot w) \qquad 1 \cdot u = 1 \qquad u \cdot 1 = 1 && (1)\\
\mathit{fail}_e \cdot u = \mathit{fail}_e && (2)\\
\textstyle\sum\{u\} = u && (3)\\
\textstyle\sum_i(\sum_j u_{ij}) = \sum_{i,j} u_{ij} \text{ (arbitrary index sets)} && (4)\\
(\textstyle\sum_i u_i) \cdot v = \sum_i u_i \cdot v \text{ (arbitrary index set)} && (5)\\
u \cdot (\textstyle\sum_i v_i) = \sum_i u \cdot v_i \text{ } (\textbf{\textit{nonempty}} \text{ index set)} && (6)\\
x \cdot (\textstyle\sum_i y_i) = \sum_i x \cdot y_i \text{ (arbitrary index set)} && (7)\\
\mathit{try}\, 1\, \mathit{catch}_e\, u = 1 && (8)\\
\mathit{try}\, \mathit{fail}_d\, \mathit{catch}_e\, u = \mathit{fail}_d, \text{ when } d \neq e && (9)\\
\mathit{try}\, \mathit{fail}_e\, \mathit{catch}_e\, u = u && (10)\\
\mathit{try}\, (\textstyle\sum_i u_i)\, \mathit{catch}_e\, w = \sum_i \mathit{try}\, u_i\, \mathit{catch}_e\, w \text{ (arbitrary index set)} && (11)\\
\mathit{try}\, (x \cdot v)\, \mathit{catch}_e\, w = x \cdot (\mathit{try}\, v\, \mathit{catch}_e\, w) && (12)
\end{aligned}$$

Lemma 2. The algebra of functions $X \to P(X \oplus (X \times E))$ is an F-quantale.

3 The Basic Algebraic Theory of Failure

In this section we investigate the basic algebraic theory of abnormal termination. One of our main goals here is to give a sound and complete axiomatization of the equational theory (in the language of KAT with fail and try-catch) of the class of relational F-quantales defined in Sect. 2. The axioms that we propose, which we call FailKAT, define a class of structures with many more interesting models other than the relational F-quantales (e.g., language models). We develop the algebraic theory of these structures. Our development consists of several steps:

- We introduce the abstract class of FailKATs. Every F-quantale (with tests), and hence every algebra of functions $X \to P(X \oplus (X \times E))$, is a model.
- We present a general model-theoretic construction that builds a FailKAT $\mathsf{F}K$ from an arbitrary KAT K. The elements of $\mathsf{F}K$ are pairs $\langle x, \psi \rangle$, where x is an element of K and $\psi : E \to K$ is an E-indexed tuple of elements of K. The component x is to be thought as the "fail-free" part, and $\psi(e)$ is the component that leads to failure with error code e.
- We show that the FailKAT $\mathsf{F}K$ can be defined equivalently in a syntactic way: expand the signature with a family of fresh constants fail_e and a family of $\mathsf{try\text{-}catch}_e$ operations, and quotient by the axioms of FailKAT.
- The aforementioned construction has several consequences, among which is the completeness of FailKAT for the theory of relational F-quantales.

Several more useful completeness results can be obtained using the results of [17,24], where free language models of KA with extra equations are identified.

$$
\begin{array}{ccc}
(x+y)+z = x+(y+z) & (x \cdot y) \cdot z = x \cdot (y \cdot z) & (x+y) \cdot z = x \cdot z + y \cdot z \\
x+y = y+x & 1 \cdot x = x & x \cdot (y+z) = x \cdot y + x \cdot z \\
x+x = x & x \cdot 1 = x & 0 \cdot x = 0 \\
x+0 = x & & x \cdot 0 = 0
\end{array}
$$

$$
1 + x \cdot x^* \leq x^* \quad 1 + x^* \cdot x \leq x^* \quad x \cdot y \leq y \Rightarrow x^* \cdot y \leq y \quad y \cdot x \leq y \Rightarrow y \cdot x^* \leq y
$$

Fig. 2. KA: axioms for Kleene algebras [12].

A *Kleene algebra* (KA) is an algebraic structure $(K, +, \cdot, {}^*, 0, 1)$ satisfying the axioms of Fig. 2, which are implicitly universally quantified. The relation $\leq$ refers to the natural partial order on K, defined as: $x \leq y \iff x + y = y$. The three top blocks of axioms (which do not involve the star operation) say that the reduct $(K, +, \cdot, 0, 1)$ is an *idempotent semiring*. We often omit the $\cdot$ operation and write xy instead of $x \cdot y$. A *Kleene algebra with tests* (KAT) is an algebraic structure $(K, B, +, \cdot, {}^*, 0, 1, \neg)$ with $B \subseteq K$, satisfying the following properties: (i) the reduct $(K, +, \cdot, {}^*, 0, 1)$ is a KA, (ii) B contains 0, 1 and is closed under $+$ and $\cdot$, and (iii) the reduct $(B, +, \cdot, 0, 1, \neg)$ is a Boolean algebra.

Definition 3 (FailKAT). Fix a set E of exceptions. A *KAT with failures* E, or simply *FailKAT*, is a three-sorted algebra

$$
(A, K, B, +, \cdot, {}^*, 0, 1, \neg, (\mathit{fail}_e)_{e \in E}, (\mathit{try\text{-}catch}_e)_{e \in E})
$$

with $B \subseteq K \subseteq A$, where K is closed under $+$, $\cdot$ and *, and $(K, B, +, \cdot, {}^*, 0, 1, \neg)$ is a KAT. Moreover, $(A, +, \cdot, {}^*, 0, 1, \mathit{fail}_e, \mathit{try\text{-}catch}_e)$ satisfies the axioms of KA except for $u \cdot 0 = 0$, and it also satisfies the *fail* axiom (2) and the *try-catch* axioms (8)–(12) of Definition 1. We say that K is the carrier of *fail-free* elements, and A is the carrier of all elements (which may allow failure).

For a KAT K and a set E of exceptions, we denote by K^E the set of E-indexed tuples (equivalently, the set of functions $E \to K$). The operation $+$ is defined on K^E componentwise: $(\phi + \psi)(e) = \phi(e) + \psi(e)$. For $x \in K$ and $\phi \in K^E$, we define $x \cdot \phi$ as follows: $(x \cdot \phi)(e) = x \cdot \phi(e)$. For all elements $x, y \in K$ and every tuple $\phi \in K^E$, the distributivity property $(x + y) \cdot \phi = (x \cdot \phi) + (y \cdot \phi)$ holds. The zero tuple $\bar{0}$ is defined as $\bar{0}(e) = 0$ for all e. For a tuple $\phi \in K^E$, an exception e, and an element $x \in K$, we write $\phi[x/e]$ for the tuple that agrees with ϕ on $E \setminus \{e\}$ and whose e-th component is equal to x. We say that a tuple is of *finite support* if it has finitely many non-zero components. We write $K^{\langle E \rangle}$ for the set of all tuples of K^E that have finite support.

Definition 4 (The Construction F). Let $(K, B, +, \cdot, ^*, 0, 1, \neg)$ be a KAT, and E a set of exceptions. We define the three-sorted algebra

$$\mathsf{F}K \triangleq (K \times K^{\langle E \rangle}, K \times \{\bar{0}\}, B \times \{\bar{0}\}, +, \cdot, ^*, 0^{\mathsf{F}}, 1^{\mathsf{F}}, \neg, \mathit{fail}_e^{\mathsf{F}}, \mathit{try\text{-}catch}_e^{\mathsf{F}})$$

with carriers $K \times K^{\langle E \rangle}$, $K \times \{\bar{0}\}$ and $B \times \{\bar{0}\}$, as follows:

$$\begin{aligned} 0^{\mathsf{F}} &\triangleq \langle 0, \bar{0} \rangle & \langle x, \phi \rangle + \langle y, \psi \rangle &\triangleq \langle x + y, \phi + \psi \rangle \\ 1^{\mathsf{F}} &\triangleq \langle 1, \bar{0} \rangle & \langle x, \phi \rangle \cdot \langle y, \psi \rangle &\triangleq \langle x \cdot y, \phi + x \cdot \psi \rangle \\ \mathit{fail}_e^{\mathsf{F}} &\triangleq \langle 0, \bar{0}[1/e] \rangle & \langle x, \phi \rangle^* &\triangleq \langle x^*, x^* \cdot \phi \rangle \\ \neg\langle p, \bar{0} \rangle &\triangleq \langle \neg p, \bar{0} \rangle & \mathit{try}\ \langle x, \phi \rangle\ \mathit{catch}_e\ \langle y, \psi \rangle &\triangleq \langle x + \phi(e) \cdot y, \phi[0/e] + \phi(e) \cdot \psi \rangle \end{aligned}$$

Informally, the idea is that an element $\langle x, \phi \rangle$ of $\mathsf{F}K$ consists of fail-free component x, and the component $\phi(e)$ which leads to failure with error code e. From the definition of $+$ we get that $\langle x, \phi \rangle \leq \langle y, \psi \rangle$ iff $x \leq y$ and $\phi(e) \leq \psi(e)$ for all e.

Definition 4 is inspired from the operational intuition of how programs with exceptions compute. Assuming that there is only one exception, we think of a pair $\langle x, \phi \rangle$ in $\mathsf{F}K$ as the program $x + \phi \cdot \mathsf{fail}$. The operation $\cdot$ in $\langle x, \phi \rangle \cdot \langle y, \psi \rangle$ models sequential composition. The fail-free component of $\langle x, \phi \rangle \cdot \langle y, \psi \rangle$ corresponds to the possibility of executing x and y in sequence, and failure can result by either executing ϕ or by executing x and ψ in sequence. The definitions of the rest of the operations can be understood similarly.

Lemma 5. Let K be a KAT and E be a set of exceptions. The algebra $\mathsf{F}K$ is a FailKAT, and the map $x \mapsto \langle x, \bar{0} \rangle$ is a KAT embedding of K into $\mathsf{F}K$.

Definition 6 (Adjoin Elements for Failure). Let $(K, B, +, \cdot, ^*, 0, 1, \neg)$ be a KAT, E be a set of exceptions, and fail_e for all $e \in E$ be fresh distinct symbols that denote failure or abnormal termination. We also consider for every exception e a fresh binary operation symbol $\mathsf{try\text{-}catch}_e$. For every element $x \in K$ we introduce a constant symbol c_x. We define the sets $\mathsf{Trm}_B(K) \subseteq \mathsf{Trm}(K) \subseteq \mathsf{Trm}_F(K)$ of algebraic terms with the following generation rules:

$$\frac{p \in B}{c_p \in \mathsf{Trm}_B(K)} \qquad \frac{s, t \in \mathsf{Trm}_B(K)}{s + t,\ s \cdot t,\ \neg s \in \mathsf{Trm}_B(K)} \qquad \frac{t \in \mathsf{Trm}_B(K)}{t \in \mathsf{Trm}(K)}$$

$$\frac{x \in K}{c_x \in \mathsf{Trm}(K)} \qquad \frac{s, t \in \mathsf{Trm}(K)}{s + t,\ s \cdot t,\ t^* \in \mathsf{Trm}(K)} \qquad \frac{t \in \mathsf{Trm}(K)}{t \in \mathsf{Trm}_F(K)}$$

$$\mathsf{fail}_e \in \mathsf{Trm}_F(K) \qquad \frac{s, t \in \mathsf{Trm}_F(K)}{s + t,\ s \cdot t,\ t^*, \mathsf{try}\, s\, \mathsf{catch}_e\, t \in \mathsf{Trm}_F(K)}$$

The function $c_x \mapsto x$, where $x \in K$, extends uniquely to a homomorphism $k : \mathsf{Trm}(K) \to K$. The *diagram* of K, defined as $\Delta_K \triangleq \{s \equiv t \mid k(s) = k(t)\}$, is the set of equations $s \equiv t$ that are true under k. In other words, Δ_K is the *kernel* of the homomorphism k. Finally, we define the set of equations

$$E_K \triangleq \text{FailKAT-Closure}(\Delta_K)$$

to be the least set that contains Δ_K and is closed under the axioms and rules of FailKAT and Horn-equational logic. By the axioms and rules of equational logic, the equations of E_K define a FailKAT-congruence on $\mathsf{Trm}_F(K)$. For a term t in $\mathsf{Trm}_F(K)$, we write $[t]_E$ to denote its congruence class. Define the three-sorted algebra $\mathbb{F}K$ with carriers $\hat{B} \subseteq \hat{K} \subseteq A$ as:

$$A \triangleq \{[t]_E \mid t \in \mathsf{Trm}_F(K)\} \quad \hat{K} \triangleq \{[t]_E \mid t \in \mathsf{Trm}(K)\} \quad \hat{B} \triangleq \{[t]_E \mid t \in \mathsf{Trm}_B(K)\}$$

Since the equations E_K define a FailKAT-congruence, we can define the FailKAT operations of $\mathbb{F}K$ on the equivalence classes of terms:

$$0^{\mathbb{F}} \triangleq [c_0]_E \qquad \mathit{fail}_e^{\mathbb{F}} \triangleq [\mathsf{fail}_e]_E \qquad [s]_E + [t]_E \triangleq [s + t]_E \qquad ([t]_E)^* \triangleq [t^*]_E$$
$$1^{\mathbb{F}} \triangleq [c_1]_E \qquad [s]_E \cdot [t]_E \triangleq [s \cdot t]_E$$

We have thus defined the algebra $\mathbb{F}K$, which has the signature of FailKATs.

Lemma 7 (Normal Form). For every term t in $\mathsf{Trm}_F(K)$ there are fail-free terms t_P and t_e in $\mathsf{Trm}(K)$ (for every exception e that appears in t) such that the equation $t \equiv t_P + \sum_e t_e \cdot \mathsf{fail}_e$ is in E_K.

Proof. Since terms are finite and only finitely many exceptions occur in them, we fix w.l.o.g. a *finite* E. The proof is by induction on the structure of t. If t is a fail-free term, then notice that the equation $t \equiv t + \sum_{e \in E} 0 \cdot \mathsf{fail}_e$ is in E_K. For the case $t = \mathsf{fail}_d$, we observe that $\mathsf{fail}_d \equiv 0 + 1 \cdot \mathsf{fail}_d + \sum_{e \neq d} 0 \cdot \mathsf{fail}_e$ is in E_K. For the induction step, we have the following equations in E_K:

$$\begin{aligned}
s + t &\equiv (s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) + (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv (s_P + t_P) + \textstyle\sum_{e \in E} (s_e + t_e) \cdot \mathsf{fail}_e \\
s \cdot t &\equiv (s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) \cdot (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv s_P \cdot t_P + \textstyle\sum_e s_P \cdot t_e \cdot \mathsf{fail}_e + \textstyle\sum_e s_e \cdot \mathsf{fail}_e \cdot (\ldots) \\
&\equiv s_P \cdot t_P + \textstyle\sum_e (s_e + s_P \cdot t_e) \cdot \mathsf{fail}_e
\end{aligned}$$

$$\begin{aligned}
t^* &\equiv (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)^* \\
&\equiv t_P^* \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e \cdot t_P^*)^* \\
&\equiv t_P^* \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e)^* \\
&\equiv t_P^* \cdot (1 + (\textstyle\sum_e t_e \cdot \mathsf{fail}_e) \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e)^*) \\
&\equiv t_P^* \cdot (1 + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv t_P^* + \textstyle\sum_e t_P^* \cdot t_e \cdot \mathsf{fail}_e \\
\mathsf{try}\, s\, \mathsf{catch}_d\, t &\equiv \mathsf{try}\,(s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e)\,\mathsf{catch}_d\,(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv \mathsf{try}\, s_P\, \mathsf{catch}_d\,(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\quad + \textstyle\sum_e \mathsf{try}\,(s_e \cdot \mathsf{fail}_e)\,\mathsf{catch}_d\,(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv s_P \cdot (\mathsf{try}\,1\,\mathsf{catch}_d\,(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)) \\
&\quad + \textstyle\sum_e s_e \cdot (\mathsf{try}\,\mathsf{fail}_e\,\mathsf{catch}_d\,(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)) \\
&\equiv s_P + s_d \cdot (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) + \textstyle\sum_{e \neq d} s_e \cdot \mathsf{fail}_e \\
&\equiv (s_P + s_d \cdot t_P) + \textstyle\sum_e s_d \cdot t_e \cdot \mathsf{fail}_e + \textstyle\sum_{e \neq d} s_e \cdot \mathsf{fail}_e \\
&\equiv (s_P + s_d \cdot t_P) + s_d \cdot t_d \cdot \mathsf{fail}_d + \textstyle\sum_{e \neq d} (s_e + s_d \cdot t_e) \cdot \mathsf{fail}_e
\end{aligned}$$

We have used the property $(x + y)^* = x^*(yx^*)^*$, which is a theorem of KA. □

Theorem 8 (F and $\mathbb{F}$). The FailKATs $\mathsf{F}K$ and $\mathbb{F}K$ are isomorphic.

Proof. We define the map $h : \mathsf{Trm}_F(K) \to \mathsf{F}K$ to be the unique homomorphism satisfying $h(c_x) = \langle x, \bar{0}\rangle$ and $h(\mathsf{fail}_e) = \langle 0, \bar{0}[1/e]\rangle$. Notice that h sends fail-free terms to fail-free elements of $\mathsf{F}K$. By Lemma 5, $\mathsf{F}K$ is a FailKAT, and therefore $h(s) = h(t)$ for every equation $s \equiv t$ in E_K. So, we can define the map $\hat{h} : \mathsf{Trm}_F(K)/E_K \to \mathsf{F}K$ by $[t]_E \mapsto h(t)$. In fact, $\hat{h}$ is a FailKAT homomorphism from $\mathbb{F}K$ to $\mathsf{F}K$. Moreover, $\hat{h}$ is surjective: for every $\langle x, \phi\rangle \in \mathsf{F}K$, where $D \subseteq E$ is the finite support of ϕ, we have that

$$\begin{aligned}
\hat{h}([c_x + \textstyle\sum_{e \in D} c_{\phi(e)} \cdot \mathsf{fail}_e]_E) &= h(c_x + \textstyle\sum_{e \in D} c_{\phi(e)} \cdot \mathsf{fail}_e) \\
&= h(c_x) + \textstyle\sum_{e \in D} h(c_{\phi(e)}) \cdot h(\mathsf{fail}_e) \\
&= \langle x, \bar{0}\rangle + \textstyle\sum_{e \in D} \langle \phi(e), \bar{0}\rangle \cdot \langle 0, \bar{0}[1/e]\rangle \\
&= \langle x, \bar{0}\rangle + \textstyle\sum_{e \in D} \langle 0, \phi(e) \cdot \bar{0}[1/e]\rangle \\
&= \langle x, \phi\rangle.
\end{aligned}$$

Finally, we claim that $\hat{h}$ is injective. Suppose that $\hat{h}([s]_E) = \hat{h}([t]_E)$. Lemma 7 says that there are fail-free terms s_P, s_e, t_P, t_e (for $e \in D \subseteq E$) in $\mathsf{Trm}(K)$ s.t.

$$s \equiv s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e \qquad\qquad t \equiv t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e$$

are equations of E_K. From our hypothesis above we obtain that

$$\begin{aligned}
\hat{h}([s]_E) = \hat{h}([t]_E) &\implies h(s) = h(t) \\
&\implies h(s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) = h(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\implies \langle k(s_P), (k(s_e))_{e \in D}\rangle = \langle k(t_P), (k(t_e))_{e \in D}\rangle \\
&\implies k(s_P) = k(t_P) \text{ and } k(s_e) = k(t_e) \text{ for all } e \in D.
\end{aligned}$$

(Recall the function $k : \mathsf{Trm}(K) \to K$ from Definition 6.) It follows that the equations $s_P \equiv t_P$ and $s_e \equiv t_e$ are in the diagram Δ_K. So, $s \equiv t$ is in E_K. We thus obtain that $[s]_E = [t]_E$. So, $\hat{h}$ is a FailKAT isomorphism. □

The above theorem says that the extension of a KAT K with fail_e elements and $\mathsf{try\text{-}catch}_e$ operations is *conservative*, since the mapping $x \mapsto \langle x, \bar{0} \rangle$ embeds K into $\mathbb{F}K \cong \mathsf{F}K$. This is a very useful property, because it means that a language of while-programs (whose semantics is defined by interpretation in a KAT) can be extended naturally to accomodate the extra programming feature of failure.

Corollary 9 (Completeness for Relational Models). FailKAT is complete for the equational theory of the class of algebras $X \to P(X \oplus (X \times E))$.

Proof. It is known from [12,13] that KA and KAT are complete for the class of relational models $X \to PX$. In fact, for a fixed finite set Σ of atomic actions and a fixed finite set B_0 of atomic tests, there is a single full relational model $\mathrm{Rel}(\Sigma, B_0)$ that characterizes the theory. We fix a finite set E of exceptions. Using the observation $P(X \oplus (X \times E)) \cong (PX) \times (PX)^E$ we can see that the algebras $X \to P(X \oplus (X \times E))$ and $\mathsf{F}(X \to PX)$ (recall Definition 4) are isomorphic. Since KAT is complete for the theory of $\mathrm{Rel}(\Sigma, B_0)$, Theorem 8 implies that FailKAT is complete for the theory of $\mathsf{F}\mathrm{Rel}(\Sigma, B_0)$. This means that FailKAT is complete for the class of all relational models. □

Let Σ be a finite set of atomic actions, and B_0 be a finite set of atomic tests. Kozen and Smith have shown in [20] that KAT is complete for $\mathsf{Reg}(\Sigma, B_0)$, the algebra of regular sets of guarded strings over Σ and B_0. Theorem 8 implies that FailKAT is complete for the algebra $\mathsf{FReg}(\Sigma, B_0)$, which is therefore the free FailKAT with action generators Σ and test generators B_0.

First, we observe that the decidability of FailKAT is an easy consequence of Lemma 7. To decide the equivalence of two terms s and t, we rewrite them according to the proof of Lemma 7 into equivalent normal forms $s \equiv s_P + \sum_e s_e \cdot \mathsf{fail}_e$ and $t \equiv t_P + \sum_e t_e \cdot \mathsf{fail}_e$, where e ranges over the exceptions that appear in s and t and all the terms s_P, s_e, t_P, t_e are fail-free. It follows that $s \equiv t$ iff $s_P \equiv t_P$ and $s_e \equiv t_e$ for all e, hence equivalence can be decided using a decision procedure for KAT. As discussed in the previous paragraph, the language model $\mathsf{FReg}(\Sigma, B_0)$ characterizes the equational theory of FailKAT, which suggests that an appropriate variant of Kozen's guarded automata [14,16] can be used to decide the equational theory in polynomial space.

Example 10. Exceptions and their handlers can be used to encode widely used constructs of non-local control flow, such as break and continue. The program

$$h = \mathsf{while}\,(\mathit{true})\;\mathsf{do}\,\{a;\;(\mathsf{if}\;\bar{p}\;\mathsf{then\,break});\;(\mathsf{if}\;\bar{q}\;\mathsf{then\,continue});\;b\}$$

can be shown to be equivalent to $h' = a;\,\mathsf{while}\,p\;\mathsf{do}\,\{(\mathsf{if}\,q\;\mathsf{then}\,b); a\}$ using FailKAT. We abbreviate $\neg p$ by $\bar{p}$, and $\neg q$ by $\bar{q}$. The program h is encoded as follows:

```
try {
    while $(true)$ do {
        try {$a$; (if $\bar{p}$ then fail$_e$); (if $\bar{q}$ then fail$_d$); $b$} catch($d$) { }
    }
} catch($e$) { }
```

Let g be the body of the while loop, and f be subprogram of the inner try-catch statement. In the language of FailKAT, we have that $f = a(\bar{p}fail_e{+}p)(\bar{q}fail_d{+}q)b$, $g = try\ f\ catch_d\, 1$, and $h = try\ (g^*0)\ catch_e\, 1$. Using the FailKAT axioms we get:

$$\begin{aligned}
f &= a\bar{p}fail_e + ap(\bar{q}fail_d + q)b = a\bar{p}fail_e + ap\bar{q}fail_d + apqb\\
g &= a\bar{p}fail_e + ap\bar{q} + apqb\\
g^* &= ((ap\bar{q} + apqb) + a\bar{p}fail_e)^* = (ap\bar{q} + apqb)^*(a\bar{p}fail_e(ap\bar{q} + apqb)^*)^*\\
&= (ap\bar{q} + apqb)^*(a\bar{p}fail_e)^* = (ap\bar{q} + apqb)^*(1 + a\bar{p}fail_e)\\
g^*0 &= (ap\bar{q} + apqb)^*(1 + a\bar{p}fail_e)0 = (ap\bar{q} + apqb)^*a\bar{p}fail_e\\
h &= (ap\bar{q} + apqb)^*a\bar{p} = (a(p\bar{q} + pqb))^*a\bar{p}\\
h' &= a(p(qb + \bar{q})a)^*\bar{p} = a((pqb + p\bar{q})a)^*\bar{p} = (a(pqb + p\bar{q}))^*a\bar{p}
\end{aligned}$$

which means that $h = h'$. We have used above the theorems $(x{+}y)^* = x^*(yx^*)^*$ and $x(yx)^* = (xy)^*x$ of KA.

Example 11. We will establish using FailKAT the equivalence of the programs of Fig. 1. The program on the right-hand side is an optimized version because it eliminates the check of the condition $i \geq 0$. To streamline the presentation we abbreviate $(i := 0)$ by a, $(X[i] := 0)$ by b, and $(i := i + 1)$ by c. We also use the abbreviations p, q and r for the tests $(i < n)$, $(0 \leq i)$ and $(i < X.\texttt{length})$ respectively, and write $\bar{p}$, $\bar{q}$, $\bar{r}$ instead of $\neg p$, $\neg q$, $\neg r$ respectively. The extra hypotheses of Fig. 1 can be then written as $a = aq$, $qb = bq$, and $qc = qcq$. Using these abbreviations we encode the right-hand side program of Fig. 1 as $R = a(p(\bar{r}fail + r)bc)^*\bar{p} = a(p\bar{r}fail + prbc)^*\bar{p} = ag^*\bar{p}$, where $g = prbc + p\bar{r}fail$, and the left-hand side program as

$$L = a(p(qr + \neg(qr)fail)bc)^*\bar{p} = a(p(qr + (\bar{q} + q\bar{r})fail)bc)^*\bar{p} = ah^*\bar{p},$$

where $h = pqrbc{+}p\bar{q}fail{+}pq\bar{r}fail$. With $qb = bq$ and $qc = qcq$ we show $qh = qhq$ and hence $qh^* = q(qhq)^* = q(qh)^* = q(pqrbc{+}pq\bar{r}fail)^* = q(q(prbc{+}p\bar{r}fail))^*$. Since $qg = qgq$ we obtain similarly that $qh^* = q(qg)^* = qg^*$. Finally, using the hypothesis $a = aq$ we obtain that $L = ah^*\bar{p} = aqh^*\bar{p} = aqg^*\bar{p} = ag^*\bar{p} = R$.

4 A Stronger Theory of Failure

In this section we investigate a stronger (i.e., with more theorems) algebraic theory of abnormal termination. We will write $\approx$ to refer to this notion of equivalence to differentiate it from the weaker equivalence that we studied in Sect. 3.

This stronger theory, called **FailTKAT**, results from FailKAT by adding the axioms

$$u = v \implies u \approx v \qquad x \cdot \mathit{fail}_e \lesssim \mathit{fail}_e \qquad \approx \text{ is KA-congruence}$$

where u, v are arbitrary elements and x is a fail-free element. As in the previous section, our ultimate goal is to give a sound and complete axiomatization of the relation $\approx$ on the algebra $X \to P(X \oplus (X \times E))$. First, we define a projection operation π that "forgets" the output state in the case or error:

$$\frac{f : X \to P(X \oplus (X \times E))}{\pi(f) : X \to P(X \oplus E)} \quad \begin{aligned} \pi(f)(x) \triangleq \{\iota_1(y) \mid \iota_1(y) \in f(x)\} \cup \\ \{\iota_2(e) \mid \iota_2(y, e) \in f(x)\} \end{aligned}$$

The $\approx$ equivalence can then be defined as follows: $f \approx g$ iff $\pi(f) = \pi(g)$.

The operation of forgetting the output state is defined for algebras of input/output relations in the obvious way, but it is not apparent if a more general construction can be formulated for a subclass of Kleene algebras. As it turns out, there exists a very natural subclass of KATs, which we call KATs with a top element, for which this is possible.

Definition 12 (KAT With Top Element). A *KAT with a top element* or a *TopKAT* is a structure $(K, B, +, \cdot, ^*, 0, 1, \neg, \top)$ so that $(K, B, +, \cdot, ^*, 0, 1, \neg)$ is a KAT and the *top element* $\top$ satisfies the inequality $x \leq \top$ for all $x \in K$.

Intuitively, the top element $\top$ is needed to forget the state of the memory. More precisely, right multiplication $(- \cdot \top)$ by the top element models the projection function that eliminates the state. Without the top element we cannot define the coarser equivalence relation $\approx$ using the operations of KA.

Definition 13 (Generalized $\approx$ Equivalence). Let K be a TopKAT and E be a set of exceptions. We define for the algebra $\mathsf{F}K$ of Definition 4 the equivalence relation $\approx$ as follows: $\langle x, \phi \rangle \approx \langle y, \psi \rangle$ iff

$$x = y \text{ and } \phi(e) \cdot \top = \psi(e) \cdot \top \text{ for every } e \in E.$$

The *projection map* $\pi : \langle x, \phi \rangle \mapsto \langle x, \phi' \rangle$ is defined as $\phi'(e) = \phi(e) \cdot \top$ for all e. So, $\approx$ can be equivalently defined as: $\langle x, \phi \rangle \approx \langle y, \psi \rangle$ iff $\pi(\langle x, \phi \rangle) = \pi(\langle y, \psi \rangle)$.

Lemma 14 (Projection). *Let K be a TopKAT and E be a set of exceptions. For the algebra $\mathsf{F}K$ and the projection map π of Definition 13 the following hold:*

$$\begin{array}{lll} \pi(\mathit{fail}_e^{\mathsf{F}}) = \langle 0, \bar{0}[\top/e] \rangle & \pi(1^{\mathsf{F}}) = 1^{\mathsf{F}} & \pi(0^{\mathsf{F}}) = 0^{\mathsf{F}} \\ \pi(u + v) = \pi(v) + \pi(v) & \pi(u \cdot v) = \pi(u) \cdot \pi(v) & \pi(u^*) = \pi(u)^* \end{array}$$

Moreover, $\pi(x \cdot \mathit{fail}_e^{\mathsf{F}} + \mathit{fail}_e^{\mathsf{F}}) = \pi(\mathit{fail}_e^{\mathsf{F}})$ or, equivalently, $x \cdot \mathit{fail}_e^{\mathsf{F}} \lesssim \mathit{fail}_e^{\mathsf{F}}$. Note: the variables u, v range over arbitrary elements and x ranges over fail-free elements.

Proof. The commutation properties are easy to verify. For the second part, we have: $\pi(\langle 0, \bar{0}[x/e]\rangle + \langle 0, \bar{0}[1/e]\rangle) = \pi(\langle 0, \bar{0}[x+1/e]\rangle) = \langle 0, \bar{0}[(x+1)\top/e]\rangle$ and also $\pi(\langle 0, \bar{0}[1/e]\rangle) = \langle 0, \bar{0}[\top/e]\rangle$. It suffices to show that $(x+1)\top = \top$ in the TopKAT K, which is true because $(x+1)\top = x\top + \top = \top$. □

The equations of Lemma 14 that show how π commutes with the KA operations of $\mathsf{F}K$ imply additionally that $\approx$ is a KA-congruence. For example, $u \approx v$ implies $\pi(u) = \pi(v)$, which gives us $\pi(u^*) = \pi(u)^* = \pi(v)^* = \pi(v^*)$ and therefore $u^* \approx v^*$. So, $\mathsf{F}K$ with $\approx$ is a model of the FailTKAT axioms.

We extend now Definition 6 to expand the algebra $\mathbb{F}K$, where K is a TopKAT, with a relation $\approx$. Similarly to the construction of the previous section, $\approx$ is given as follows for terms s and t: $[s]_E \approx [t]_E$ iff $s \approx t$ is provable using the system FailTKAT and the diagram of K.

Theorem 15. *Let K be a TopKAT and E be a set of exceptions. The FailTKATs $(\mathsf{F}K, \approx)$ and $(\mathbb{F}K, \approx)$ are isomorphic.*

Proof. The proof extends the one for Theorem 8. It remains to show that for all terms s and t: $[s]_E \approx [t]_E$ iff $\hat{h}([s]_E) \approx \hat{h}([t]_E)$. The right-to-left direction is the interesting one. By Lemma 7, we bring s and t to their normal forms and:

$$\begin{aligned} \hat{h}([s]_E) \approx \hat{h}([t]_E) &\implies \pi(h(s)) = \pi(h(t)) \\ &\implies \pi(h(s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e)) = \pi(h(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)) \\ &\implies \langle k(s_P), (k(s_e)\top)_{e \in D}\rangle = \langle k(t_P), (k(t_e)\top)_{e \in D}\rangle \\ &\implies k(s_P) = k(t_P) \text{ and } k(s_e \cdot c_\top) = k(t_e \cdot c_\top) \text{ for all } e \in D. \end{aligned}$$

It follows that $s_P \equiv t_P$ and $s_e \cdot c_\top \equiv t_e \cdot c_\top$ are in the diagram of K. Now,

$$\begin{aligned} s &\approx s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e \approx s_P + \textstyle\sum_e s_e \cdot c_\top \cdot \mathsf{fail}_e \\ &\approx t_P + \textstyle\sum_e t_e \cdot c_\top \cdot \mathsf{fail}_e \approx t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e \approx t \end{aligned}$$

is provable, because $x \cdot \mathsf{fail}_e \approx x \cdot c_\top \cdot \mathsf{fail}_e$ follows from Δ_K and FailTKAT.

$$1 \lesssim c_\top \Rightarrow x \cdot \mathsf{fail}_e \lesssim x \cdot c_\top \cdot \mathsf{fail}_e \qquad c_\top \cdot \mathsf{fail}_e \lesssim \mathsf{fail}_e \Rightarrow x \cdot c_\top \cdot \mathsf{fail}_e \lesssim x \cdot \mathsf{fail}_e$$

We thus obtain that $[s]_E \approx [t]_E$. So, $\hat{h}$ is a FailTKAT isomorphism. □

Similarly to Theorem 8 of the previous section, we interpret Theorem 15 as saying that an arbitrary KAT with a top element can be *extended conservatively* into a KAT with failure that satisfies the additional axiom $x \cdot \mathit{fail}_e \lesssim \mathit{fail}_e$. The mapping $x \mapsto \langle x, \bar{0}\rangle$ embeds the TopKAT K into the extension $(\mathsf{F}K, \approx)$.

Corollary 16 (Completeness for Relational Models). *FailTKAT is complete for the theory of the relation $\approx$ on the class of algebras $X \to P(X \oplus (X \times E))$. Notation: Recall that $f \approx g$ iff the projections of f and g to functions of type $X \to P(X \oplus E)$ (by applying π) are equal.*

Proof (sketch). Similar to the proof of Corollary 9. We have to show that the first-order structures $X \to P(X \oplus (X \times E))$ and $\mathsf{F}(X \to PX)$ (signature extended with the projection π and the equivalence $\approx$) are isomorphic. This relies on the observation $P(X \oplus E) \cong (PX) \times (\mathbb{1} \oplus \mathbb{1})^E \cong (PX) \times (\{\emptyset\} \oplus \{X\})^E$. $\square$

In the previous section we discussed how our conservativity result for FailKAT gives as easy consequences the existence of a free language model, the decidability of the theory, and also suggests a way to approach the question of complexity using guarded automata. The situation is similar for FailTKAT, it suffices to observe that the free KAT $\mathsf{Reg}(\Sigma, B_0)$ with action generators Σ and test generators B_0 has a top element: the guarded language denoted by the expression Σ^*.

5 Related Work

As far as the basic theory of failure is concerned, the works [9,15] are closely related to ours. In both these papers, extensions of KAT are investigated that can be used for reasoning about nonlocal flow of control, using e.g. labels and goto statements. Syntactically, these systems amount essentially to using matrices of expressions, where the row index corresponds to an entry label and the column index corresponds to an exit label. While the fail operation can be encoded using such general constructs of nonlocal flow of control, the works [9,15] do not address the question of whether it is possible to axiomatize fail directly, i.e. without translation into a more complicated language. We have shown here that this is indeed the case, which is a new result and does not follow from any of [9,15]. The system FailKAT that we introduce here axiomatizes the properties of failure directly, and thus offers a more convenient style of reasoning for this computational phenomenon. More importantly, we ***depart completely*** from the investigations of [9,15] when we consider the stronger theory FailTKAT. None of these earlier systems can capture the properties of failure under the coarser equivalence that we study here.

Aceto and Hennessy study in [1,2] a process algebra that includes an explicit symbol δ for *deadlock*. This is somewhat similar to our fail operation, since δ satisfies the equational property $\delta; x = \delta$. Our work is, however, markedly distinct and contributes very different results. The work of Aceto and Hennessy studies a notion of bisimulation preorder, which does not have the same theory as language or trace equivalence. We axiomatize here the input-output behavior of programs, which correponds to language (and not bisimulation) equivalence.

The literature on computational effects, monads [8,28], and algebraic operations [11,30,31] is somewhat related to our work at a conceptual level. Within this body of work, exceptions and failure are modeled using the formalism of monads. In sharp contrast to what we are doing here, the work on monads typically focuses on the type structure (whereas we have here only two sorts!) and different program structuring operations (e.g., products and function abstraction) in the setting of a functional language. At a technical level, there is not much of an intersection between our investigations and the work on monads. The language of KAT is generally more abstract than the monad-based formalisms, which maybe include constructs like products, as in $A \times B \to P(A \times B)$, or return values, as in $A \to (S \to P(S \times B))$. Such extra constructs can easily make it impossible to obtain any kind of useful completeness theorem. In particular, if the language allows loops and binary products, then we can encode abstractions of imperative programs whose state can be decomposed in variables that can be read from and written to independently. This is the case of the so-called "two-variable while program schemes", whose partial-correctness and equational theory are *not recursively enumerable* [22]. This suggests that the abstraction level of KA/KAT is necessary for obtaining meaningful unconditional completeness theorems for programs with an iteration construct.

6 Conclusion

We have considered here two algebraic theories, called FailKAT and FailTKAT, for imperative while programs with an explicit fail operation that causes abnormal termination. The system FailKAT captures the notion of program equivalence that results from a semantics that allows for the observation of the final state upon failure. The system FailTKAT captures a coarser notion of equivalence, namely when we cannot observe the final state of the computation upon failure. Both notions of equivalence are meaningful and useful, and we have seen that they admit simple and intuitive axiomatizations. From a technical perspective, the case of FailTKAT is more challenging and interesting.

A important direction for future work is the study of FailKAT and FailTKAT in the coalgebraic setting. Such an investigation would contribute to the question (posed by Kozen in [15]) of whether there is a simple coalgebraic treatment of nonlocal flow of control involving a definition of derivatives [4,5] for the nonlocal control flow constructs. We expect that the *fail* and *try-catch* constructs, which are much more structured than labels and goto statements, lend themselves to an elegant coalgebraic treatment. At a practical level, this would give rise to simple and efficient automata-theoretic decision procedures.

Another interesting question is whether the ideas of the present paper can be applied to other logical systems. Some apparent candidates are variations of KAT such as NetKAT [3,7] and Nominal KA [18,19]. Abnormal termination and nonlocal flow of control have been studied in the context of partial correctness theories based on Hoare logic (see, for example, [6,10,29,33,34]). It seems likely that the axioms of FailKAT can inspire axioms and rules for Hoare logics that treat failure and exception handling, and even obtain unconditional completeness results (see, for example, [21,23,25–27]) in the propositional setting.

References

1. Aceto, L., Hennessy, M.: Termination, deadlock and divergence. In: Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1989. LNCS, vol. 442, pp. 301–318. Springer, New York (1990). doi:10.1007/BFb0040264
2. Aceto, L., Hennessy, M.: Termination, deadlock, and divergence. J. ACM **39**(1), 147–187 (1992)
3. Anderson, C.J., Foster, N., Guha, A., Jeannin, J.B., Kozen, D., Schlesinger, C., Walker, D.: NetKAT: semantic foundations for networks. In: Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014), pp. 113–126 (2014)
4. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2), 291–319 (1996)
5. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
6. Delbianco, G.A., Nanevski, A.: Hoare-style reasoning with (algebraic) continuations. In: Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP 2013), pp. 363–376 (2013)
7. Foster, N., Kozen, D., Mamouras, K., Reitblatt, M., Silva, A.: Probabilistic NetKAT. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 282–309. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49498-1_12
8. Goncharov, S., Schröder, L., Mossakowski, T.: Kleene monads: handling iteration in a framework of generic effects. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 18–33. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03741-2_3
9. Grathwohl, N.B.B., Kozen, D., Mamouras, K.: KAT + B! In: Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, pp. 44:1–44:10 (2014)
10. Huisman, M., Jacobs, B.: Java program verification via a Hoare logic with abrupt termination. In: Maibaum, T. (ed.) FASE 2000. LNCS, vol. 1783, pp. 284–303. Springer, Heidelberg (2000). doi:10.1007/3-540-46428-X_20
11. Hyland, M., Plotkin, G., Power, J.: Combining effects: sum and tensor. Theor. Comput. Sci. **357**(1), 70–99 (2006)
12. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Inf. Comput. **110**(2), 366–390 (1994)
13. Kozen, D.: Kleene algebra with tests. Trans. Programm. Lang. Syst. **19**(3), 427–443 (1997)
14. Kozen, D.: Automata on guarded strings and applications. Matématica Contemporânea **24**, 117–139 (2003)
15. Kozen, D.: Nonlocal flow of control and Kleene algebra with tests. In: Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008), pp. 105–117 (2008)
16. Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. Technical report, Computing and Information Science, Cornell University, March 2008
17. Kozen, D., Mamouras, K.: Kleene algebra with equations. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 280–292. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43951-7_24
18. Kozen, D., Mamouras, K., Petrişan, D., Silva, A.: Nominal Kleene coalgebra. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 286–298. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_23

19. Kozen, D., Mamouras, K., Silva, A.: Completeness and incompleteness in nominal Kleene algebra. In: Kahl, W., Winter, M., Oliveira, J.N. (eds.) RAMICS 2015. LNCS, vol. 9348, pp. 51–66. Springer, Cham (2015). doi:10.1007/978-3-319-24704-5_4
20. Kozen, D., Smith, F.: Kleene algebra with tests: completeness and decidability. In: Dalen, D., Bezem, M. (eds.) CSL 1996. LNCS, vol. 1258, pp. 244–259. Springer, Heidelberg (1997). doi:10.1007/3-540-63172-0_43
21. Kozen, D., Tiuryn, J.: On the completeness of propositional Hoare logic. Inf. Sci. **139**(3–4), 187–195 (2001)
22. Luckham, D.C., Park, D.M.R., Paterson, M.S.: On formalised computer programs. J. Comput. Syst. Sci. **4**(3), 220–249 (1970)
23. Mamouras, K.: On the Hoare theory of monadic recursion schemes. In: Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, pp. 69:1–69:10 (2014)
24. Mamouras, K.: Extensions of Kleene algebra for program verification. Ph.D. thesis, Cornell University, Ithaca, NY, August 2015
25. Mamouras, K.: Synthesis of strategies and the Hoare logic of angelic nondeterminism. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 25–40. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46678-0_2
26. Mamouras, K.: The Hoare logic of deterministic and nondeterministic monadic recursion schemes. ACM Trans. Comput. Logic (TOCL) **17**(2), 13:1–13:30 (2016)
27. Mamouras, K.: Synthesis of strategies using the Hoare logic of angelic and demonic nondeterminism. Log. Methods Comput. Sci. **12**(3), 1–41 (2016)
28. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991)
29. von Oheimb, D.: Hoare logic for Java in Isabelle/HOL. Concurr. Comput. Pract. Exp. **13**(13), 1173–1214 (2001)
30. Plotkin, G., Power, J.: Computational effects and operations. ENTCS **73**, 149–163 (2004)
31. Plotkin, G., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00590-9_7
32. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (FOCS 1976), pp. 109–121 (1976)
33. Saabas, A., Uustalu, T.: A compositional natural semantics and Hoare logic for low-level languages. Theor. Comput. Sci. **373**(3), 273–302 (2007)
34. Tan, G., Appel, A.W.: A compositional logic for control flow. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 80–94. Springer, Heidelberg (2005). doi:10.1007/11609773_6