A call-by-value λ-calculus with lists and control

Robbert Krebbers

Radboud University Nijmegen

`mail@robbertkrebbers.nl`

Calculi with control operators have been studied to reason about control in programming languages and to interpret the computational content of classical proofs. To make these calculi into a real programming language, one should also include data types.

As a step into that direction, this paper defines a simply typed call-by-value λ-calculus with the control operators `catch` and `throw`, a data type of lists, and an operator for primitive recursion (à la Gödel's **T**). We prove that our system satisfies subject reduction, progress, confluence for untyped terms, and strong normalization for well-typed terms.

1 Introduction

The extension of simply typed λ-calculus with control operators and the observation that these operators can be typed using rules of classical logic is originally due to Griffin [Gri90] and has lead to a lot of research by varying the control operators, the underlying calculus or the computation rules, or by studying concrete examples of the computational content of classical proofs. Little of this research has considered the problem of how to incorporate primitive data types in direct style. If one wants to use these calculi as a real functional programming language with control, this is a gap that needs filling.

This paper contributes towards the development of a λ-calculus with both data types and control operators that allows program extraction from classical proofs. In such a calculus one can write specifications of programs, which can be proven using (a restricted form of) classical logic. Program extraction would then allow to extract a program from such a proof where the classical reasoning steps are extracted to control operators. This approach yields programs-with-control that are *correct by construction* because they are extracted from a proof of the specification. However, in order for these extracted programs to be useful in practice, data types in direct style should be supported.

As a step into that direction, we introduce $\lambda::\texttt{catch}$, a simply typed call-by-value λ-calculus with the control operators `catch` and `throw`, a list and unit data type, and an operator for primitive recursion (à la Gödel's **T**). We consider lists because those are among the most commonly used data types in functional programming. Expressively, lists make our system as least as strong as Gödel's **T** because natural numbers can be encoded as lists over the unit type. We prove the conventional meta theoretical properties – subject reduction, progress, confluence, and strong normalization – so that it may be used as a sound basis for a calculus that allows program extraction from classical proofs.

Our system is based on Herbelin's IQC_{MP}-calculus with `catch` and `throw` that he uses to give a computational interpretation of Markov's principle [Her10]. Most importantly, we adopt his restriction of the control operator `catch` to $\rightarrow$-free types. This restriction enables the system to satisfy *progress* without losing other meta theoretical properties. The progress property states that if t is a well-typed closed term, then t is either a value or there is a term t' such that t reduces to t'. From a programmer's point of view this is an important property as together with confluence it ensures *unique representation of data*. For example, for the natural numbers, unique representation of data means that for each natural number there is (up to conversion) a unique closed term of the type of natural numbers. To show how

the system can be used in programming, we give a simple example in 2.11, where we define a function that multiplies the values of a list and throws an exception as soon as it encounters the value 0.

Proving confluence or strong normalization for systems with control generally requires complex extensions of standard proof methods, see for example [Par97, Py98, BHF01, Nak03, GKM12, RS94]. For λ::catch this is less the case. We give relatively short proofs of subject reduction, progress, confluence for untyped terms, and strong normalization for well-typed terms.

1.1   Related work

Incorporating data types into a λ-calculus with control has not received much attention. We briefly summarize the research done in this direction and compare it with our work.

Parigot [Par92] has described a variant of his $\lambda\mu$-calculus with second-order types. His system is very powerful, because all the well-known second-order representable data types are included in it. But as observed in [Par92, Par93], it does not ensure unique representation of data. This defect can be remedied by adding additional reduction rules, however, this results in a loss of confluence. Another approach is to use output operators to extract data, but this introduces an additional indirection.

Rehof and Sørensen have described an extension of their λ_Δ-calculus with basic constants and functions [RS94]. Unfortunately their extension is quite limited. In particular, an operator for primitive recursion, which takes terms rather than basic constants as its arguments, cannot be defined.

Barthe and Uustalu [BU02] have considered CPS-translations for inductive and coinductive types. In particular, they describe a system with a primitive for iteration over the natural numbers, and the control operator Δ. They prove preservation of typing and reduction under a CPS-translation, but do not consider other meta theoretical properties of this system.

Crolard and Polonowski [CP11] have considered a version of Gödel's **T** with products and call/cc. However, as their semantics is presented by CPS-translations instead of a direct specification via a calculus, their work is not directly related to ours.

Geuvers, Krebbers and McKinna [GKM12] have defined an extension of Parigot's $\lambda\mu$-calculus with a data type of natural numbers and an operator for primitive recursion. They prove that their system satisfies subject reduction, unique representation of the naturals, confluence and strong normalization. Also, they define a CPS-translation into Gödel's **T** to show that adding control operators does not extend the expressive power. Unfortunately, their system is call-by-name with call-by-value evaluation for data types, making it less suitable to model control in most programming languages. Due to their decision to use $\lambda\mu$, their proofs involve many complex extensions of standard proof techniques, and expose a lot of non-trivial interaction between control and data types.

Several extensions of λ-calculus with the control operators catch and throw have been studied in the literature. We discuss those that are most relevant to our work. Crolard [Cro99] has considered a call-by-name variant of such a calculus, for which he defines a correspondence with Parigot's $\lambda\mu$-calculus. He uses this correspondence to prove confluence, subject reduction and strong normalization, but does not consider data types in direct style.

Herbelin [Her10] has defined IQC_{MP}, a calculus with catch and throw to give a computational interpretation of Markov's principle. His calculus is call-by-value and supports product, sum, existential, and universally quantified types. An essential feature of his calculus is the restriction of catch to $\forall$-$\rightarrow$-free types. This restriction enables him to prove progress, which is an important property for his main result, a proof of the disjunction and existence property.

Since Herbelin's IQC_{MP}-calculus has a convenient meta theory, we use it as the starting point for our work. But instead of considering product, sum, existential, and universally quantified types, we consider

a data type of lists in direct style. Whereas Herbelin does not consider confluence, and does not give a direct proof of strong normalization, we will give direct proofs of these properties for our system.

1.2 Outline

In Section 2, we define the typing rules, and the basic reduction rules, whose compatible closure defines computation in λ::catch. We give two example programs showing interaction between data types and control. Section 2 moreover contains proofs of subject reduction and progress. Section 3 contains a direct proof of confluence for untyped terms based on an analysis of complete developments. Section 4 contains a direct proof of strong normalization using the reducibility method. We close with conclusions and indications for further work in Section 5.

2 The system

Definition 2.1. *The* types, terms *and* values *of λ::catch are defined as*

$$\sigma, \tau, \rho ::= \top \mid [\tau] \mid \sigma \to \tau$$
$$t, r, s ::= x \mid () \mid \texttt{nil} \mid (::) \mid \texttt{lrec} \mid \lambda x.r \mid ts \mid \texttt{catch}\,\alpha.t \mid \texttt{throw}\,\alpha\,t$$
$$v, w, v_r, v_s ::= x \mid () \mid \texttt{nil} \mid (::) \mid (::)\,v \mid (::)\,v\,w \mid \texttt{lrec} \mid \texttt{lrec}\,v_r \mid \texttt{lrec}\,v_r\,v_s \mid \lambda x.r$$

where x, y, and z range over variables, *and α, β and γ range over* continuation variables.

The construct $\lambda x.r$ binds x in r, and catch $\alpha.t$ binds α in t. The precedence of λ and catch is lower than application, so instead of catch $\alpha.(tr)$ we write catch $\alpha.tr$. We let $\text{FV}(t)$ denote the set of free variables of t, and $\text{FCV}(t)$ the set of free continuation variables of t. As usual, we use *Barendregt's variable convention* [Bar84]. That is, given a term, we may assume that bound variables are distinct from free variables and that all bound variables are distinct. The operation of capture avoiding substitution $t[x := r]$ of r for x in t is defined in the usual way.

The constructs nil and (::) are the constructors of the list data type. We treat these constructors, and the operator lrec for primitive recursion over lists, as unary constants so we can use them in partially applied position. Also, this treatment results in a more uniform definition of the reduction rules. We often use Haskell-style notation. In particular, we write $t :: r$ to denote $(::)\,t\,r$, and $\lambda\_.t$ to denote $\lambda x.t$ with $x \notin \text{FV}(t)$. Furthermore, we write $[t_1, \ldots, t_n]$ to denote $t_1 :: \ldots :: t_n :: \texttt{nil}$.

Following Herbelin [Her10] we restrict catch to $\to$-free types. Without this restriction, progress (Theorem 2.15) would fail. Let us consider the term catch $\alpha.\lambda x.$throw $\alpha\,(\lambda y.y)$. Without this restriction, this term would have had type $\top \to \top$, whereas it would not reduce to a value. In fact, even (catch $\alpha.\lambda x.$throw $\alpha\,(\lambda y.y)$) () : $\top$ would not reduce. The reduction rules for catch and throw are very similar to [Her10], but quite different from those by Crolard [Cro99]. In particular, Crolard includes reduction rules to move the catch whereas Herbelin's system and ours merely allow a throw to move towards the corresponding catch. This is due to the restriction to $\to$-free types.

Definition 2.2. *We let ϕ and ψ range over $\to$-free types.*

Definition 2.3. *Let Γ be a map from variables to types, and let Δ be a map from continuation variables to $\to$-free types. The derivation rules for the typing judgment $\Gamma; \Delta \vdash t : \rho$ are as shown below.*

$$\frac{x : \rho \in \Gamma}{\Gamma; \Delta \vdash x : \rho} \qquad \frac{}{\Gamma; \Delta \vdash () : \top} \qquad \frac{}{\Gamma; \Delta \vdash \texttt{nil} : [\sigma]} \qquad \frac{}{\Gamma; \Delta \vdash (::) : \sigma \to [\sigma] \to [\sigma]}$$

$$\overline{\Gamma;\Delta \vdash \texttt{lrec} : \rho \to (\sigma \to [\sigma] \to \rho \to \rho) \to [\sigma] \to \rho}$$

$$\frac{\Gamma, x:\sigma;\Delta \vdash t:\tau}{\Gamma;\Delta \vdash \lambda x.t : \sigma \to \tau} \qquad \frac{\Gamma;\Delta \vdash t:\sigma \to \tau \qquad \Gamma;\Delta \vdash s:\sigma}{\Gamma;\Delta \vdash ts:\tau}$$

$$\frac{\Gamma;\Delta, \alpha:\psi \vdash t:\psi}{\Gamma;\Delta \vdash \texttt{catch}\,\alpha.t:\psi} \qquad \frac{\Gamma;\Delta \vdash t:\psi \qquad \alpha:\psi \in \Delta}{\Gamma;\Delta \vdash \texttt{throw}\,\alpha\,t:\tau}$$

Lemma 2.4. *Given a value v with $;\Delta \vdash v:\rho$, then:*

1. *If $\rho = \top$, then v is of the shape $()$.*

2. *If $\rho = [\sigma]$, then v is of the shape $[w_1,\ldots,w_n]$.*

3. *If $\rho = \sigma \to \tau$, then v is of the shape $(::)$, $(::)w$, $\texttt{lrec}$, $\texttt{lrec}\,v_r$, $\texttt{lrec}\,v_r\,v_s$ or $\lambda x.r$.*

Proof. This result is proven by induction on the structure of v. The case $v \equiv x$ is impossible because v is closed for free variables. The other cases are easy. □

Definition 2.5. *The* contexts *of λ::catch *are defined as:*

$$E ::= \Box t \mid v\Box \mid \texttt{throw}\,\alpha\,\Box$$

Given a context E and a term s, the substitution *of s for the hole in E, notation $E[s]$, is defined in the usual way.*

Definition 2.6. *Reduction $t \to t'$ is defined as the compatible closure of:*

$$(\lambda x.t)\,v \to t[x := v] \tag{β_v}$$
$$E[\texttt{throw}\,\alpha\,t] \to \texttt{throw}\,\alpha\,t \tag{t}$$
$$\texttt{catch}\,\alpha.\texttt{throw}\,\alpha\,t \to \texttt{catch}\,\alpha.t \tag{c1}$$
$$\texttt{catch}\,\alpha.\texttt{throw}\,\beta\,v \to \texttt{throw}\,\beta\,v \text{ if } \alpha \notin \{\beta\} \cup \mathrm{FCV}(v) \tag{c2}$$
$$\texttt{catch}\,\alpha.v \to v \qquad \text{if } \alpha \notin \mathrm{FCV}(v) \tag{c3}$$
$$\texttt{lrec}\,v_r\,v_s\,\texttt{nil} \to v_r \tag{nil}$$
$$\texttt{lrec}\,v_r\,v_s\,(v_h :: v_t) \to v_s\,v_h\,v_t\,(\texttt{lrec}\,v_r\,v_s\,v_t) \tag{::}$$

As usual, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.

Notice that because we treat partially applied $(::)$ and $\texttt{lrec}$ constructs as values, we get reductions like $\texttt{throw}\,\alpha\,r :: t \equiv (::)\,(\texttt{throw}\,\alpha\,r)\,t \to (\texttt{throw}\,\alpha\,r)\,t \to \texttt{throw}\,\alpha\,r$ for free without the need for additional contexts for $(::)$ and $\texttt{lrec}$.

Fact 2.7. *If $\Gamma;\Delta \vdash v:\psi$, then $\mathrm{FCV}(v) = \emptyset$*

Proof. By induction on the structure of the value v. Since ψ is $\to$-free, we only have to consider the cases $v \equiv x$, $v \equiv ()$, $v \equiv \texttt{nil}$ and $v \equiv v_l :: v_r$, for which the result trivially holds. □

The reduction rules (c2) and (c3) require that $\alpha \notin \mathrm{FCV}(v)$. This side condition can be omitted for well-typed terms by the previous fact. However, since we consider the problem of confluence for untyped terms (Section 3), we do need this additional restriction.

Definition 2.8. *We define a type for the natural numbers* $\mathbb{N} := [\top]$, *with the following operations on it.*

$$0 := \mathtt{nil}$$
$$\mathtt{S} := (::)\,()$$
$$\mathtt{nrec} := \lambda x_r x_s\,.\,\mathtt{lrec}\,x_r\,(\lambda\,\_.x_s)$$

We let $\underline{n} := \mathtt{S}^n 0$ *denote the representation of a natural number.*

Fact 2.9. *The operations on* $\mathbb{N}$ *satisfy the expected conversions.*

$$\mathtt{nrec}\,v_r\,v_s\,0 \twoheadrightarrow v_r$$
$$\mathtt{nrec}\,v_r\,v_s\,(\mathtt{S}\,v) = v_s\,v\,(\mathtt{nrec}\,v_r\,v_s\,v)$$

Colson and Fredholm [CF98] have shown that in Gödel's **T** with call-by-value reduction, it takes at least a number of steps that is linear with respect to the input for a non-trivial algorithm to reduce to a value. In particular, it is impossible to compute the predecessor in constant time. Intuitively it is easy to see why, consider the reduction $\mathtt{nrec}\,v_r\,v_s\,(\mathtt{S}\,v) \to v_s\,v\,(\mathtt{nrec}\,v_r\,v_s\,v)$. Due to the restriction of β-reduction to values, the recursive call, $\mathtt{nrec}\,v_r\,v_s\,v$ has to be reduced to a value before the whole term is able to reduce to a value. In λ::catch we can use the control mechanism to do better.

Example 2.10. *We define the predecessor function* $\mathtt{pred} : \mathbb{N} \to \mathbb{N}$ *as follows.*

$$\mathtt{pred} := \lambda n\,.\,\mathtt{catch}\,\alpha\,.\,\mathtt{nrec}\,0\,(\lambda x\,.\,\mathtt{throw}\,\alpha\,x)\,n$$

Computing the predecessor is possible in a constant number of steps.

$$\mathtt{pred}\,\underline{n+1} \twoheadrightarrow \mathtt{catch}\,\alpha\,.\,\mathtt{nrec}\,0\,(\lambda x\,.\,\mathtt{throw}\,\alpha\,x)\,(\mathtt{S}\,\underline{n})$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,(\lambda x\,.\,\mathtt{throw}\,\alpha\,x)\,\underline{n}\,(\mathtt{lrec}\,0\,(\lambda\,\_x\,.\,\mathtt{throw}\,\alpha\,x)\,\underline{n})$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,(\mathtt{throw}\,\alpha\,\underline{n})\,(\mathtt{lrec}\,0\,(\lambda\,\_x\,.\,\mathtt{throw}\,\alpha\,x)\,\underline{n})$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,\mathtt{throw}\,\alpha\,\underline{n} \twoheadrightarrow \underline{n}$$

Example 2.11. *We define a* λ::catch-*program* $F : [\mathbb{N}] \to \mathbb{N}$ *that computes the product of the elements of a list. The interest of this program is that it uses the control mechanism to stop multiplying once the value 0 is encountered.*

$$F := \lambda l\,.\,\mathtt{catch}\,\alpha\,.\,\mathtt{lrec}\,\underline{1}\,H\,l$$
$$H := \lambda x\,\_.\,\mathtt{nrec}\,(\mathtt{throw}\,\alpha\,0)\,(\lambda y\,\_h\,.\,\mathtt{S}\,y * h)\,x$$

Here, addition $(+)$ *and multiplication* $(*)$ *are defined as follows.*

$$(+) := \lambda nm\,.\,\mathtt{nrec}\,m\,(\lambda\,\_\,y\,.\,\mathtt{S}\,y)\,n$$
$$(*) := \lambda nm\,.\,\mathtt{nrec}\,0\,(\lambda\,\_\,y\,.\,m+y)\,n$$

We show a computation of $F\,[\underline{4},\underline{0},\underline{9}]$.

$$F\,[\underline{4},\underline{0},\underline{9}] \twoheadrightarrow \mathtt{catch}\,\alpha\,.\,\mathtt{lrec}\,\underline{1}\,H\,[\underline{4},\underline{0},\underline{9}]$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,\mathtt{nrec}\,(\mathtt{throw}\,\alpha\,0)\,(\lambda y\,\_h\,.\,\mathtt{S}\,y * h)\,\underline{4}\,(\mathtt{lrec}\,\underline{1}\,H\,[\underline{0},\underline{9}])$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,(\lambda h\,.\,\underline{4} * h)\,(\mathtt{lrec}\,\underline{1}\,H\,[\underline{0},\underline{9}])$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,(\lambda h\,.\,\underline{4} * h)\,(\mathtt{throw}\,\alpha\,0)$$
$$\twoheadrightarrow \mathtt{catch}\,\alpha\,.\,\mathtt{throw}\,\alpha\,0 \twoheadrightarrow 0$$

Lemma 2.12. *If* $\Gamma;\Delta \vdash r : \sigma$ *and* $\Gamma,x : \sigma;\Delta \vdash t : \rho$*, then* $\Gamma;\Delta \vdash t[x := r] : \rho$*.*

Theorem 2.13 (Subject reduction)**.** *If* $\Gamma;\Delta \vdash t : \rho$ *and* $t \rightarrow t'$*, then* $\Gamma;\Delta \vdash t' : \rho$*.*

Proof. We have to show that each reduction rule preserves typing. We use Lemma 2.12 for (β_v). □

Lemma 2.14. *Given a normal form* t *with* $;\Delta \vdash t : \rho$*, then either* t *is a value, or* $t \equiv$ throw β v *for some value* v *and continuation variable* β*.*

Proof. This result is proven by induction on the derivation of $;\Delta \vdash t : \rho$.

1. Let $;\Delta \vdash x : \rho$ with $x : \rho \in \emptyset$. This is impossible because $x : \rho \notin \emptyset$.

2. In the case of $()$, nil, $(::)$, lrec and $\lambda x.r$ the result is immediate.

3. Let $;\Delta \vdash ts : \tau$ with $;\Delta \vdash t : \sigma \rightarrow \tau$ and $;\Delta \vdash s : \sigma$. By the induction hypothesis we know that the terms r and s are either a value or a throw. Since ts is in normal form, it is impossible that either of them is a throw. Therefore, we may assume that both are values. Now, since t has type $\sigma \rightarrow \tau$, we can use Lemma 2.4 to analyze the possible shapes of t.

   (a) Let $t \equiv$ lrec $v_r v_s$. By the typing rules we obtain that s has type $[\rho]$ for some ρ. So, by Lemma 2.4 we have that s is a list. However, ts is in normal form, so this is impossible.

   (b) Let $t \equiv \lambda x.r$. This case is impossible because s is a value and ts is in normal form.

   (c) In all other cases, the term ts is a value.

4. Let $;\Delta \vdash$ catch $\alpha.t : \psi$ with $;\Delta, \alpha : \psi \vdash t : \psi$. By the induction hypothesis we know that t is a value or a throw. If t is a value, Fact 2.7 gives us that $\alpha \notin \mathrm{FCV}(t)$. This is impossible since catch $\alpha.t$ is in normal form. Similarly, it is also impossible that t is a throw.

5. Let $;\Delta \vdash$ throw α $t : \sigma$ with $;\Delta \vdash t : \psi$ and $\alpha : \psi \in \Delta$. By the induction hypothesis we know that t is a value or a throw. If t is a value, we are done. Furthermore, t cannot be a throw since throw α t is in normal form. □

Theorem 2.15 (Progress)**.** *If* $; \vdash t : \rho$*, then* t *is either a value, or there is a term* t' *with* $t \rightarrow t'$*.*

Proof. This result follows immediately from Lemma 2.14. □

3   Confluence

To prove confluence for untyped terms of λ::catch, we use the notion of *parallel reduction*, as introduced by Tait and Martin-Löf [Bar84]. A parallel reduction relation $\Rightarrow$ allows to contract a number of redexes in a term simultaneously so as to make it being preserved under substitution. If one proves that the parallel reduction $\Rightarrow$ satisfies:

- The *diamond property*: if $t_1 \Rightarrow t_2$ and $t_1 \Rightarrow t_3$, then there exists a t_4 such that $t_2 \Rightarrow t_4$ and $t_3 \Rightarrow t_4$.

- $t_1 \Rightarrow t_2$ implies $t_1 \twoheadrightarrow t_2$ and $t_1 \twoheadrightarrow t_2$ implies $t_1 \Rightarrow^* t_2$.

then one obtains confluence of $\rightarrow$.

Following Takahashi [Tak95], we further streamline the proof by defining the *complete development* of a term t, notation $t^\diamond$, which is obtained by contracting all redexes in t. Now to prove the diamond property of $\Rightarrow$, it suffices to prove that $t_1 \Rightarrow t_2$ implies $t_2 \Rightarrow t_1^\diamond$.

For Parigot's $\lambda\mu$-calculus, it is well known that the naive parallel reduction is not preserved under substitution [BHF01]. Instead, a complex parallel reduction that moves subterms located very deeply in

a term towards the outside is needed [BHF01, Nak03, GKM12]. For λ::catch we experience another issue. Consider the following rule.

$$\text{If } t \Rightarrow t', \text{ then } E[\text{throw } \alpha \, t] \Rightarrow \text{throw } \alpha \, t'$$

If we take throw α_1 (throw α_2 (... throw α_n ()...)) (with $n \geq 5$), then we could perform a reduction that contracts all even numbered throws, and also a reduction that contracts all odd numbered throws. Since these two reducts do not converge in a single parallel reduction step, such a parallel reduction would not be confluent. To repair this issue we use a similar fix as in [BHF01, Nak03, GKM12]: we allow a throw to jump over a *compound context*.

Definition 3.1. Compound contexts *are defined as:*

$$\vec{E} ::= \Box \mid \vec{E}t \mid v\vec{E} \mid \text{throw } \alpha \, \vec{E}$$

Given a compound context $\vec{E}$ and a term s, the substitution of s for the hole in $\vec{E}$, *notation $\vec{E}[s]$, is defined in the usual way.*

Definition 3.2. Parallel reduction $t \Rightarrow t'$ *is inductively defined as:*

1. $x \Rightarrow x$, $() \Rightarrow ()$, $\text{nil} \Rightarrow \text{nil}$, $(::) \Rightarrow (::)$, *and* $\text{nrec} \Rightarrow \text{nrec}$.

2. *If* $t \Rightarrow t'$ *and* $r \Rightarrow r'$, *then* $tr \Rightarrow t'r'$.

3. *If* $t \Rightarrow t'$, *then* $\lambda x.t \Rightarrow \lambda x.t'$.

4. *If* $t \Rightarrow t'$, *then* $\text{catch } \alpha .t \Rightarrow \text{catch } \alpha .t'$.

5. *If* $t \Rightarrow t'$ *and* $v \Rightarrow r$, *then* $(\lambda x.t) v \Rightarrow t'[x := r]$.

6. *If* $t \Rightarrow t'$, *then* $\vec{E}[\text{throw } \alpha \, t] \Rightarrow \text{throw } \alpha \, t'$.

7. *If* $t \Rightarrow t'$, *then* $\text{catch } \alpha .\text{throw } \alpha \, t \Rightarrow \text{catch } \alpha .t'$.

8. *If* $v \Rightarrow t$ *and* $\alpha \notin \{\beta\} \cup \text{FCV}(v)$, *then* $\text{catch } \alpha .\text{throw } \beta \, v \Rightarrow \text{throw } \beta \, t$.

9. *If* $v \Rightarrow t$ *and* $\alpha \notin \text{FV}(v)$, *then* $\text{catch } \alpha .v \Rightarrow t$.

10. *If* $v_r \Rightarrow r$, *then* $\text{lrec } v_r \, v_s \, \text{nil} \Rightarrow r$.

11. *If* $v_r \Rightarrow r$, $v_s \Rightarrow s$, $v_h \Rightarrow h$ *and* $v_t \Rightarrow t$, *then* $\text{lrec } v_r \, v_s \, (v_h :: v_t) \Rightarrow s \, h \, t \, (\text{lrec } r \, s \, t)$.

Lemma 3.3. *Parallel reduction satisfies the following properties.*

1. *It is reflexive, i.e.* $t \Rightarrow t$.

2. *The term* $v[x := w]$ *is a value.*

3. *If* $v \Rightarrow t$, *then t is a value.*

4. *If* $t \Rightarrow t'$, *then* $\text{FV}(t') \subseteq \text{FV}(t)$ *and* $\text{FCV}(t') \subseteq \text{FCV}(t)$.

5. *If* $t \Rightarrow t'$ *and* $v \Rightarrow r$, *then* $t[x := v] \Rightarrow t'[x := r]$.

Lemma 3.4. *Parallel reduction enjoys the intended behavior. That is:*

1. *If* $t \rightarrow t'$, *then* $t \Rightarrow t'$.

2. *If* $t \Rightarrow t'$, *then* $t \twoheadrightarrow t'$.

Proof. The first property is proven by induction on the derivation of $t \rightarrow t'$ using that parallel reduction is reflexive and satisfies the substitution property (Lemma 3.3). The second property is proven by induction on the derivation of $t \Rightarrow t'$ using an obvious substitution lemma for $\twoheadrightarrow$. $\qquad\square$

Definition 3.5. *The* complete development $t^\diamond$ *is defined as:*

$$((\lambda x.t)\,v)^\diamond := t^\diamond[x := v^\diamond]$$
$$(\vec{E}[\texttt{throw}\ \alpha\ t])^\diamond := \texttt{throw}\ \alpha\ t^\diamond \qquad \text{if } t \not\equiv \texttt{throw}\ \gamma\ s$$
$$(\texttt{catch}\ \alpha.\texttt{throw}\ \alpha\ t)^\diamond := \texttt{catch}\ \alpha.t^\diamond$$
$$(\texttt{catch}\ \alpha.\texttt{throw}\ \beta\ v)^\diamond := \texttt{throw}\ \beta\ v^\diamond \qquad \text{if } \alpha \notin \{\beta\} \cup \text{FCV}(v)$$
$$(\texttt{catch}\ \alpha.v)^\diamond := v^\diamond \qquad\qquad\quad \text{if } \alpha \notin \text{FCV}(v)$$
$$(\texttt{lrec}\ v_r\ v_s\ \texttt{nil})^\diamond := v_r^\diamond$$
$$(\texttt{lrec}\ v_r\ v_s\ (v_h :: v_t))^\diamond := v_s^\diamond\ v_h^\diamond\ v_t^\diamond\ (\texttt{lrec}\ v_r^\diamond\ v_s^\diamond\ v_t^\diamond)$$

For variables, (), nil, (::) *and* nrec, *the complete development is defined as the identity, and it propagates through the other cases that we have omitted.*

We lift the parallel reduction $\Rightarrow$ to compound contexts with the intended behavior that if $\vec{E} \Rightarrow \vec{F}$ and $q \Rightarrow q'$, then $\vec{E}[\texttt{throw}\ \alpha\ q] \Rightarrow \vec{F}[\texttt{throw}\ \alpha\ q']$.

Definition 3.6. Parallel reduction $\vec{E} \Rightarrow \vec{F}$ on compound contexts is inductively defined as:

1. $\square \Rightarrow \square$

2. $\texttt{throw}\ \alpha\ \square \Rightarrow \square$

3. *If* $\vec{E} \Rightarrow \vec{F}$ *and* $t \Rightarrow t'$, *then* $\vec{E}t \Rightarrow \vec{F}t'$.

4. *If* $\vec{E} \Rightarrow \vec{F}$ *and* $v \Rightarrow t$, *then* $v\vec{E} \Rightarrow t\vec{F}$.

5. *If* $\vec{E} \Rightarrow \vec{F}$, *then* $\texttt{throw}\ \alpha\ \vec{E} \Rightarrow \texttt{throw}\ \alpha\ \vec{F}$.

6. *If* $\vec{E} \Rightarrow \vec{F}$, *then* $\texttt{throw}\ \beta\ (\texttt{throw}\ \alpha\ \vec{E}) \Rightarrow \texttt{throw}\ \alpha\ \vec{F}$.

Remark that if we have that $\vec{E}[\texttt{throw}\ \alpha\ q] \Rightarrow r$, then r is not necessarily of the shape $\vec{F}[\texttt{throw}\ \alpha\ q']$ with $\vec{E} \Rightarrow \vec{F}$ and $q \Rightarrow q'$ because q could be a throw.

Lemma 3.7. *If* $\vec{E}[\texttt{throw}\ \alpha\ q_1] \Rightarrow r$ *and* $q_1 \not\equiv \texttt{throw}\ \gamma\ s$, *then there exists a* q_2 *and* $\vec{F}$ *such that* $r \equiv \vec{F}[\texttt{throw}\ \alpha\ q_2]$ *with* $\vec{E} \Rightarrow \vec{F}$ *and* $q_1 \Rightarrow q_2$.

Lemma 3.8. *If* $t_1 \Rightarrow t_2$, *then* $t_2 \Rightarrow t_1^\diamond$.

Proof. By induction on the derivation of $t_1 \Rightarrow t_2$. We consider some interesting cases.

1. Let $t_1\,r_1 \Rightarrow t_2\,r_2$ with $t_1 \Rightarrow t_2$ and $r_1 \Rightarrow r_2$. We distinguish the following cases:

   (a) Let $t_1 \equiv \lambda x.s_1$ and r_1 a value. By distinguishing reductions we have $t_2 \equiv \lambda x.s_2$ with $s_1 \Rightarrow s_2$. Now, $t_2 \Rightarrow t_1^\diamond$ and $s_2 \Rightarrow s_1^\diamond$ by the induction hypothesis. Furthermore, we have that r_2 is a value by Lemma 3.3. Therefore, $t_2\,r_2 \equiv (\lambda x.s_2)\,r_2 \Rightarrow s_1^\diamond[x := r_1^\diamond] \equiv (t_1\,r_1)^\diamond$ by Lemma 3.3.

   (b) Let $t_1 \equiv \texttt{nrec}\ v_r\ v_s$ and $r_1 \equiv \texttt{nil}$. By distinguishing reductions we have $t_2 \equiv \texttt{nrec}\ r\ s$ and $r_2 \equiv \texttt{nil}$ with $v_r \Rightarrow r$ and $v_s \Rightarrow s$. Now, $r \Rightarrow v_r^\diamond$ by the induction hypothesis. Therefore, $t_2\,r_2 \equiv \texttt{nrec}\ r\ s\ \texttt{nil} \Rightarrow v_r^\diamond \equiv (\texttt{nrec}\ v_r\ v_s\ \texttt{nil})^\diamond \equiv (t_1\,r_1)^\diamond$.

   (c) Let $t_1 \equiv \texttt{nrec}\ v_r\ v_s$ and $r_1 \equiv v_h :: v_t$. This case is similar to the previous one.

   (d) Let $t_1 \equiv \vec{E}[\texttt{throw}\ \beta\ q_1]$ with $q_1 \not\equiv \texttt{throw}\ \gamma\ s$. By Lemma 3.7, we have $t_2 \equiv \vec{F}[\texttt{throw}\ \alpha\ q_2]$ with $\vec{E} \Rightarrow \vec{F}$ and $q_1 \Rightarrow q_2$. Now we have $q_2 \Rightarrow q_1^\diamond$ by the induction hypothesis. Therefore, $t_2\,r_2 \equiv \vec{F}[\texttt{throw}\ \alpha\ q_2]\,r_1 \Rightarrow \texttt{throw}\ \alpha\ q_1^\diamond \equiv (t_1\,r_1)^\diamond$.

   (e) Let $r_1 \equiv \vec{E}[\texttt{throw}\ \beta\ q_1]$ with $q_1 \not\equiv \texttt{throw}\ \gamma\ s$ and t_1 a value. This proof of this case is similar to the previous one.

(f) For the remaining cases we have $t_2 \Rightarrow t_1^\diamond$ and $r_2 \Rightarrow r_1^\diamond$ by the induction hypothesis. Therefore, $t_2\,r_2 \Rightarrow t_1^\diamond r_1^\diamond \equiv (t_1\,r_1)^\diamond$.

2. Let $\mathtt{catch}\,\alpha.t_1 \Rightarrow \mathtt{catch}\,\alpha.t_2$ with $t_1 \Rightarrow t_2$. We distinguish the following cases:

   (a) Let $t_1 \equiv \mathtt{throw}\,\alpha\,q_1$ with $q_1 \not\equiv \mathtt{throw}\,\gamma\,s$. By distinguishing reductions we obtain that $t_2 \equiv \mathtt{throw}\,\alpha\,q_2$ with $q_1 \Rightarrow q_2$. Now we have $q_2 \Rightarrow q_1^\diamond$ by the induction hypothesis. Therefore, $\mathtt{catch}\,\alpha.t_2 \equiv \mathtt{catch}\,\alpha.\mathtt{throw}\,\alpha\,q_2 \Rightarrow \mathtt{catch}\,\alpha.q_1^\diamond \equiv (\mathtt{catch}\,\alpha.t_1)^\diamond$.

   (b) Let $t_1 \equiv \mathtt{throw}\,\alpha\,(\vec{E}[\mathtt{throw}\,\beta\,q_1])$ with $q_1 \not\equiv \mathtt{throw}\,\gamma\,s$. We have $t_2 \equiv \vec{F}[\mathtt{throw}\,\beta\,q_2]$ with $\mathtt{throw}\,\alpha\,\vec{E} \Rightarrow \vec{F}$ and $q_1 \Rightarrow q_2$ by Lemma 3.7. Also, $q_2 \Rightarrow q_1^\diamond$ by the induction hypothesis. Therefore, $\mathtt{catch}\,\alpha.t_1 \equiv \mathtt{catch}\,\alpha.\vec{F}[\mathtt{throw}\,\beta\,q_2] \Rightarrow \mathtt{catch}\,\alpha.q_1^\diamond \equiv (\mathtt{catch}\,\alpha.t_1)^\diamond$.

   (c) Let $t_1 \equiv \mathtt{throw}\,\beta\,v_1$ with $\alpha \notin \{\beta\} \cup \mathrm{FV}(v_1)$. By distinguishing reductions we obtain that $t_2 \equiv \mathtt{throw}\,\beta\,v_2$ with $v_1 \Rightarrow v_2$. Now, $v_2 \Rightarrow v_1^\diamond$ by the induction hypothesis, and $\alpha \notin \mathrm{FCV}(v_2)$ by Lemma 3.3. So, $\mathtt{catch}\,\alpha.t_2 \equiv \mathtt{catch}\,\alpha.\mathtt{throw}\,\beta\,v_2 \Rightarrow \mathtt{throw}\,\beta\,v_1^\diamond \equiv (\mathtt{catch}\,\alpha.t_1)^\diamond$.

   (d) Let t_1 be a value with $\alpha \notin \mathrm{FCV}(t_1)$. We have $t_2 \Rightarrow t_1^\diamond$ by the induction hypothesis. Also, t_2 is a value and $\alpha \notin \mathrm{FCV}(t_2)$ by Lemma 3.3. Therefore, $\mathtt{catch}\,\alpha.t_2 \Rightarrow t_1^\diamond \equiv (\mathtt{catch}\,\alpha.t_1)^\diamond$.

   (e) For the remaining cases we have $t_2 \Rightarrow t_1^\diamond$ by the induction hypothesis. As a result we have $\mathtt{catch}\,\alpha.t_2 \Rightarrow \mathtt{catch}\,\alpha.t_1^\diamond \equiv (\mathtt{catch}\,\alpha.t_1)^\diamond$.

3. Let $\vec{E}[\mathtt{throw}\,\alpha\,t_1] \Rightarrow \mathtt{throw}\,\alpha\,t_2$ with $t_1 \Rightarrow t_2$. We distinguish the following cases:

   (a) Let $t_1 \equiv \vec{E}[\mathtt{throw}\,\beta\,q_1]$ with $q_1 \not\equiv \mathtt{throw}\,\gamma\,s$. This case is similar to 1d.

   (b) For the remaining cases we have $t_2 \Rightarrow t_1^\diamond$ by the induction hypothesis. As a result we have $\mathtt{throw}\,\alpha\,t_2 \Rightarrow \mathtt{throw}\,\alpha\,t_1^\diamond \equiv (\vec{E}[\mathtt{throw}\,\alpha\,t_1])^\diamond$.

4. Let $\mathtt{catch}\,\alpha.\mathtt{throw}\,\alpha\,t_1 \Rightarrow \mathtt{catch}\,\alpha.t_2$ with $t_1 \Rightarrow t_2$. We have $t_2 \Rightarrow t_1^\diamond$ by the induction hypothesis. As a result we have $\mathtt{catch}\,\alpha.t_2 \Rightarrow \mathtt{catch}\,\alpha.t_1^\diamond \equiv (\mathtt{catch}\,\alpha.\mathtt{throw}\,\alpha\,t_1)^\diamond$.

5. Let $\mathtt{catch}\,\alpha.\mathtt{throw}\,\beta\,v_1 \Rightarrow \mathtt{throw}\,\beta\,t_2$ with $v_1 \Rightarrow t_2$, $\alpha \notin \{\beta\} \cup \mathrm{FV}(v_1)$. We have $t_2 \Rightarrow v_1^\diamond$ by the induction hypothesis. Furthermore, t_2 is a value by Lemma 3.3. As a result we have $\mathtt{throw}\,\beta\,t_2 \Rightarrow \mathtt{throw}\,\beta\,v_1^\diamond \equiv (\mathtt{catch}\,\alpha.\mathtt{throw}\,\beta\,v_1)^\diamond$.

6. Let $\mathtt{catch}\,\alpha.v_1 \Rightarrow t_2$ with $v_1 \Rightarrow t_2$ and $\alpha \notin \mathrm{FV}(v_1)$. We have $t_2 \Rightarrow v_1^\diamond$ by the induction hypothesis and t_2 is a value by Lemma 3.3. Therefore, $t_2 \Rightarrow v_1^\diamond \equiv (\mathtt{catch}\,\alpha.v_1)^\diamond$. $\qquad\square$

Corollary 3.9. *If $t_1 \Rightarrow t_2$ and $t_1 \Rightarrow t_3$, then there exists a t_4 such that $t_2 \Rightarrow t_4$ and $t_3 \Rightarrow t_4$.*

Proof. Take $t_4 := t_1^\diamond$. Now we have $t_2 \Rightarrow t_1^\diamond$ and $t_3 \Rightarrow t_1^\diamond$ by Lemma 3.8. $\qquad\square$

Theorem 3.10 (Confluence). *If $t_1 \twoheadrightarrow t_2$ and $t_1 \twoheadrightarrow t_3$, then there exists a t_4 such that $t_2 \twoheadrightarrow t_4$ and $t_3 \twoheadrightarrow t_4$.*

Proof. By Corollary 3.9 and a simple diagram chase (as in [Bar84]), we obtain confluence of $\Rightarrow$. Now, confluence of $\to$ is immediate by Lemma 3.4. $\qquad\square$

4 Strong normalization

In this section we prove that reduction in $\lambda{::}\mathtt{catch}$ is strongly normalizing. We use the reducibility method, which is originally due to Tait [Tai67]. By this method, instead of proving that a term t of type ρ is strongly normalizing, one proves $t \in [\![\rho]\!]$, where $[\![\sigma \to \tau]\!] := \{t \mid \forall s \in [\![\sigma]\!] \,.\, ts \in [\![\tau]\!]\}$.

Although Tait's method does work for the call-by-name $\lambda\mu$-calculus [Par97], David and Nour [DN05] have shown that it does not extend to its symmetric variant. They proved that the property, if $r \in \mathtt{SN}$ and

$t[x := r] \in [\![\sigma]\!]$, then $(\lambda x.t)\, r \in [\![\sigma]\!]$, no longer holds due to the reduction $t\,(\mu\alpha.c) \to \mu\alpha.c[\alpha := \alpha(t\Box)]$. However, the similar reduction $t\,(\mathtt{throw}\,\alpha\,r) \to \mathtt{throw}\,\alpha\,r$ in our calculus consumes t without performing any (structural) substitution in r. So, for λ::catch this problem does not exist.

It may be possible to prove strong normalization by use of a strictly reduction preserving translation into another system that is already known to be strongly normalizing. For example, one may try to use the obvious translation into the second-order call-by-value $\lambda\mu$-calculus where the data type of lists can be defined as $[\tau] := \forall X.\, X \to (\tau \to X \to X) \to X$. However, this translation does not preserve the reduction (::). We are unaware of other systems that are both known to be strongly normalizing, and allow a straightforward strictly reduction preserving translation.

Definition 4.1. *The set of* strongly normalizing terms*, SN, contains the terms t for which the length of each reduction sequence starting at t is bounded. We use the notation $\nu(t)$ to denote this bound.*

Due to the addition of lists to λ::catch, the interpretation becomes a bit more involved than for the case of $\lambda\!\to$. Intuitively, we want our interpretation to ensure that each element of the list $t \in [\![[\sigma]]\!]$ is contained in $[\![\sigma]\!]$.

Definition 4.2. *Given a set of terms S, the set of terms $\mathcal{L}_S$ is inductively defined by the following rule.*

$$\frac{\forall v\,w\,.\, if\, t \twoheadrightarrow v :: w\ then\ v \in S\ and\ w \in \mathcal{L}_S}{t \in \mathcal{L}_S}$$

Notice that the above definition ensures that $\mathtt{nil} \in \mathcal{L}_S$ because $\mathtt{nil}$ cannot reduce to $v :: w$.

Definition 4.3. *The interpretation $[\![\rho]\!]$ of a type ρ is defined as:*

$$[\![\top]\!] := \mathtt{SN}$$
$$[\![[\sigma]]\!] := \mathtt{SN} \cap \mathcal{L}_{[\![\sigma]\!]}$$
$$[\![\sigma \to \tau]\!] := \{t \mid \forall s \in [\![\sigma]\!]\,.\, ts \in [\![\tau]\!]\}$$

Lemma 4.5 and 4.8 establish an important property: $[\![\psi]\!] = \mathtt{SN}$ for $\to$-free types ψ. Since the catch operator is restricted to $\to$-free types, this means that $\mathtt{catch}\,\alpha.r \in \mathtt{SN}$ implies $\mathtt{catch}\,\alpha.r \in [\![\psi]\!]$. This property is the key result to prove that $r \in [\![\psi]\!]$ implies $\mathtt{catch}\,\alpha.r \in [\![\psi]\!]$ (Lemma 4.15).

The property $r \in [\![\sigma]\!]$ implies $\mathtt{catch}\,\alpha.r \in [\![\sigma]\!]$ does not hold for all types σ. For example, consider $t \equiv (\mathtt{catch}\,\alpha.\mathtt{throw}\,\alpha\,\omega)\,\omega$ with $\omega = \lambda x.xx$. By Corollary 4.10 we have $\mathtt{throw}\,\alpha\,\omega \in [\![\top \to \top]\!]$ and using the above result we would have had $t \in \mathtt{SN}$. This is impossible because $t \twoheadrightarrow \omega\omega \to \omega\omega \to \ldots$

Definition 4.4. *We define the* size *of t, notation $\ell(t)$, as the number of symbols in t. For $t \in \mathtt{SN}$, we define $\ell_n(t)$ as the size of the normal form of t.*

Lemma 4.5. *If ψ is $\to$-free, then $\mathtt{SN} \subseteq [\![\psi]\!]$.*

Proof. We have to show that for each $t \in \mathtt{SN}$, we have $t \in [\![\psi]\!]$. We proceed by well-founded induction on $\ell_n(t)$ and a case distinction on the structure of ψ. The only interesting case is (list), where we have to show that $t \in \mathcal{L}_{[\![\psi]\!]}$. So, let $t \twoheadrightarrow v :: w$ for values v and w. We have $v \in \mathtt{SN} \subseteq [\![\psi]\!]$ and $w \in [\![[\psi]]\!]$ by the induction hypothesis as $\ell_n(v) < \ell_n(t)$ and $\ell_n(w) < \ell_n(t)$. Hence, $t \in \mathcal{L}_{[\![\psi]\!]}$ as required. ☐

Lemma 4.6. *If $t \in [\![\sigma]\!]$ and $t \twoheadrightarrow t'$, then $t' \in [\![\sigma]\!]$.*

Proof. We prove this result by structural induction on σ.

(unit) Let $t \in [\![\top]\!] = \mathtt{SN}$ and $t \twoheadrightarrow t'$. By definition of $\mathtt{SN}$ we have $t' \in \mathtt{SN}$.

(list) Let $t \in [[\sigma]] = \mathtt{SN} \cap \mathscr{L}_{[\![\sigma]\!]}$ and $t \twoheadrightarrow t'$. As we have $t' \in \mathtt{SN}$ by definition of $\mathtt{SN}$, it remains to prove that $t' \in \mathscr{L}_{[\![\sigma]\!]}$. So, let $t' \twoheadrightarrow v :: w$ for values v and w. Now we have $t \twoheadrightarrow t' \twoheadrightarrow v :: w$. Therefore, $v \in [\![\sigma]\!]$ and $w \in \mathscr{L}_{[\![\sigma]\!]}$ by the assumption that $t \in \mathscr{L}_{[\![\sigma]\!]}$.

($\rightarrow$) Let $t \in [\![\sigma \rightarrow \tau]\!]$ and $t \twoheadrightarrow t'$. Since we have to prove that $t' \in [\![\sigma \rightarrow \tau]\!]$, let $r \in [\![\sigma]\!]$. By assumption we have $tr \in [\![\tau]\!]$. Furthermore we have $tr \twoheadrightarrow t'r$ because $t \twoheadrightarrow t'$. Therefore, $t'r \in [\![\tau]\!]$ by the induction hypothesis. □

Definition 4.7. *We let $\vec{t}$ and $\vec{u}$ denote a sequence of terms. The set $\overrightarrow{\mathtt{SN}}$ contains all sequences of strongly normalizing terms.*

Lemma 4.8. *We have the following results:*

1. $[\![\sigma]\!] \subseteq \mathtt{SN}$.
2. *If $\vec{u} \in \overrightarrow{\mathtt{SN}}$ then $x\vec{u} \in [\![\sigma]\!]$.*

Proof. The results are proven simultaneously by structural induction on σ.

(unit) Both results are immediate.

(list) Property (1). $[[\sigma]] = \mathtt{SN} \cap \mathscr{L}_{[\![\sigma]\!]} \subseteq \mathtt{SN}$.

Property (2). Let $\vec{u} \in \overrightarrow{\mathtt{SN}}$. We have to show that $x\vec{u} \in [[\sigma]] = \mathtt{SN} \cap \mathscr{L}_{[\![\sigma]\!]}$. Since it is immediate that $x\vec{u} \in \mathtt{SN}$, it remains to show that $x\vec{u} \in \mathscr{L}_{[\![\sigma]\!]}$. However, as reductions $x\vec{u} \twoheadrightarrow v :: w$ are impossible, we are done.

($\rightarrow$) Property (1). Let $t \in [\![\sigma \rightarrow \tau]\!]$. We have $x \in [\![\sigma]\!]$ by the induction hypothesis of property (2), and therefore $tx \in [\![\tau]\!]$. By the induction hypothesis of property (1) we have $[\![\tau]\!] \subseteq \mathtt{SN}$, so $t \in \mathtt{SN}$.

Property (2). Let $\vec{u} \in \overrightarrow{\mathtt{SN}}$. We have to show that $x\vec{u} \in [\![\sigma \rightarrow \tau]\!]$, so let $r \in [\![\sigma]\!]$. By the induction hypothesis of property (1) we have $r \in \mathtt{SN}$, and therefore $x\vec{u}r \in [\![\tau]\!]$ by the induction hypothesis of property (2). Therefore, $x\vec{u} \in [\![\sigma \rightarrow \tau]\!]$ as required. □

Lemma 4.9. *If $r \in \mathtt{SN}$ and $\vec{u} \in \overrightarrow{\mathtt{SN}}$, then $(\mathtt{throw}\ \alpha\ r)\vec{u} \in \mathtt{SN}$.*

Proof. We prove this result by induction on the length of $\vec{u}$.

1. We prove that we have $\mathtt{throw}\ \alpha\ r \in \mathtt{SN}$ by induction on $\nu(r)$. We proceed by distinguishing the reductions $\mathtt{throw}\ \alpha\ r \rightarrow q$ and show that we have $q \in \mathtt{SN}$ for each such a q.

   (a) Let $\mathtt{throw}\ \alpha\ (\mathtt{throw}\ \beta\ t) \rightarrow \mathtt{throw}\ \beta\ t$. The result holds by assumption.
   (b) Let $\mathtt{throw}\ \alpha\ r \rightarrow \mathtt{throw}\ \alpha\ r'$ with $r \rightarrow r'$. The result follows from the induction hypothesis.

2. We prove that we have $(\mathtt{throw}\ \alpha\ r)t\,\vec{u} \in \mathtt{SN}$ by induction on $\nu(t) + \nu((\mathtt{throw}\ \alpha\ r)\vec{u})$. It is easy to verify that $q \in \mathtt{SN}$ for all reductions $(\mathtt{throw}\ \alpha\ r)t\,\vec{u} \rightarrow q$. □

Corollary 4.10. *If $r \in \mathtt{SN}$ and $\vec{u} \in \overrightarrow{\mathtt{SN}}$, then $(\mathtt{throw}\ \alpha\ r)\vec{u} \in [\![\sigma]\!]$.*

Proof. We prove this result by structural induction on σ.

(unit) This case is a direct consequence of Lemma 4.9.

(list) We have to show that $(\mathtt{throw}\ \alpha\ r)\vec{u} \in [[\sigma]] = \mathtt{SN} \cap \mathscr{L}_{[\![\sigma]\!]}$. As we have $(\mathtt{throw}\ \alpha\ r)\vec{u} \in \mathtt{SN}$ by Lemma 4.9, it remains to show that $(\mathtt{throw}\ \alpha\ r)\vec{u} \in \mathscr{L}_{[\![\sigma]\!]}$. So, let $(\mathtt{throw}\ \alpha\ r)\vec{u} \twoheadrightarrow v :: w$ for values v and w. By distinguishing reductions we see that this reduction is impossible.

($\rightarrow$) This case follows directly from the induction hypothesis and Lemma 4.8. □

It would be convenient if we could prove $t \in [\![\sigma]\!]$ by showing that for all reductions $t \to t'$ we have $t' \in [\![\sigma]\!]$. Unfortunately, this result does not hold in general. For example, whereas the term $\omega :: \mathtt{nil}$ is in normal form, we do not have $\omega :: \mathtt{nil} \in [\![[() \to ()]]\!]$. Similarly to Girard *et al.* [GTL89], we restrict ourselves to the terms t that are *neutral*.

Definition 4.11. *A term is* neutral *if it is not of the shape $\lambda x.r$, $\mathtt{nrec}\ v_r\ v_s$, or $v :: w$.*

Lemma 4.12. *If t is neutral, and for all terms t' with $t \to t'$ we have $t' \in [\![\sigma]\!]$, then $t \in [\![\sigma]\!]$.*

Proof. The results is proven by structural induction on σ.

(unit) The result is immediate.

(list) Let t be a neutral term such that for all terms t' with $t \to t'$ we have $t' \in [\![[\sigma]]\!]$. We have to prove that $t \in [\![[\sigma]]\!] = \mathtt{SN} \cap \mathscr{L}_{[\![\sigma]\!]}$. By Lemma 4.8 we have $[\![[\sigma]]\!] \subseteq \mathtt{SN}$, and therefore $t \in \mathtt{SN}$ as $t' \in \mathtt{SN}$ for each t' with $t \to t'$ by assumption. It remains to show that $t \in \mathscr{L}_{[\![\sigma]\!]}$, so let $t \twoheadrightarrow v :: w$ for values v and w. Since t is neutral, there should be a term t' such that $t \to t' \twoheadrightarrow v :: w$. For such a term t' we have $t' \in [\![[\sigma]]\!]$ by assumption, hence $v \in [\![\sigma]\!]$ and $w \in \mathscr{L}_{[\![\sigma]\!]}$. Therefore, $t \in \mathscr{L}_{[\![\sigma]\!]}$ as required.

($\to$) Let t be a neutral term such that for all terms t' with $t \to t'$ we have $t' \in [\![\sigma \to \tau]\!]$. We have to prove that $t \in [\![\sigma \to \tau]\!]$, so let $r \in [\![\sigma]\!]$. By the induction hypothesis it is sufficient to show that if $tr \to q$ then $q \in [\![\tau]\!]$. By Lemma 4.8 we have $r \in \mathtt{SN}$, so we proceed by induction on $\nu(r)$. We distinguish the following reductions.

(a) Let $tr \to t'r$ with $t \to t'$. Now we have $t' \in [\![\sigma \to \tau]\!]$ by assumption. Hence, $t'r \in [\![\tau]\!]$ by definition, so we are done.

(b) Let $tr \to tr'$ with $r \to r'$. The result follows from the induction hypothesis.

(c) Let $(\mathtt{throw}\ \alpha\ s)\ r \to \mathtt{throw}\ \alpha\ s$. By Lemma 4.8 we have $[\![\sigma \to \tau]\!] \subseteq \mathtt{SN}$, and therefore $\mathtt{throw}\ \alpha\ s \in \mathtt{SN}$ as $t' \in \mathtt{SN}$ for each t' with $\mathtt{throw}\ \alpha\ s \to t'$ by assumption. As a consequence we have $\mathtt{throw}\ \alpha\ s \in [\![\tau]\!]$ by Corollary 4.10.

(d) Let $v\ (\mathtt{throw}\ \alpha\ s) \to \mathtt{throw}\ \alpha\ s$. By assumption we have $\mathtt{throw}\ \alpha\ s \in [\![\sigma]\!]$, so $\mathtt{throw}\ \alpha\ s \in \mathtt{SN}$ by Lemma 4.8. Hence, $\mathtt{throw}\ \alpha\ s \in [\![\tau]\!]$ by Corollary 4.10.

No other reductions are possible because t is neutral (so, in particular it cannot be of the shape $\lambda x.s$ or $\mathtt{nrec}\ v_r\ v_s$). □

Lemma 4.13. *If $r \in \mathtt{SN}$ and $t[x := r] \in [\![\sigma]\!]$, then $(\lambda x.t)\ r \in [\![\sigma]\!]$.*

Proof. We prove this result by well-founded induction on $\nu(t) + \nu(r)$. By Lemma 4.12 it is sufficient to show that for each q with $(\lambda x.t)\ r \to q$ we have $q \in [\![\sigma]\!]$. We consider some interesting reductions.

1. Let $(\lambda x.t)\ v \to t[x := v]$. The result holds by assumption.

2. Let $(\lambda x.t)\ (\mathtt{throw}\ \beta\ r) \to \mathtt{throw}\ \beta\ r$. In this case we have $\mathtt{throw}\ \beta\ r \in [\![\sigma]\!]$ by Corollary 4.10. □

Lemma 4.14. *If $t \in [\![\sigma]\!]$ and $s \in [\![[\sigma]]\!]$, then $t :: s \in [\![[\sigma]]\!]$.*

Proof. First we have to prove that $t :: s \in \mathtt{SN}$. That means, for each q with $t :: s \to q$ we have $q \in \mathtt{SN}$. We prove this result by induction on $\nu(t) + \nu(s)$. We consider the following reductions.

1. Let $\mathtt{throw}\ \alpha\ r :: s \to (\mathtt{throw}\ \alpha\ r)\ s$. Since we have $\mathtt{throw}\ \alpha\ r \in [\![\sigma]\!]$ and $s \in [\![[\sigma]]\!]$ by assumption, we obtain that $r, s \in \mathtt{SN}$ by Lemma 4.8. Therefore, $(\mathtt{throw}\ \alpha\ r)\ s \in \mathtt{SN}$ by Lemma 4.9.

2. Let $v :: \mathtt{throw}\ \alpha\ r \to \mathtt{throw}\ \alpha\ r$. Since we have $\mathtt{throw}\ \alpha\ r \in [\![[\sigma]]\!]$ by assumption, we obtain that $\mathtt{throw}\ \alpha\ r \in \mathtt{SN}$ by Lemma 4.8.

Secondly, we have to prove that $t :: s \in \mathscr{L}_{[\![\sigma]\!]}$. So, let $t :: s \twoheadrightarrow v :: w$ for values v and w. By distinguishing reductions we obtain that $t \twoheadrightarrow v$ and $s \twoheadrightarrow w$. Therefore, we have $v \in [\![\sigma]\!]$ and $w \in \mathscr{L}_{[\![\sigma]\!]}$ by Lemma 4.6. Hence, $t :: s \in \mathscr{L}_{[\![\sigma]\!]}$ as required. □

Lemma 4.15. *If ψ is $\rightarrow$-free and $r \in [\![\psi]\!]$, then* $\mathtt{catch}\,\alpha\,.\,r \in [\![\psi]\!]$.

Proof. By Lemma 4.5 it is sufficient to prove that $\mathtt{catch}\,\alpha\,.\,r \in \mathtt{SN}$. We prove this result by well-founded induction on the lexicographic order on $\nu(r)$ and $\ell(r)$. Let q with $\mathtt{catch}\,\alpha\,.\,r \to q$. It remains to prove prove that $q \in \mathtt{SN}$. We consider the following interesting reductions.

1. Let $\mathtt{catch}\,\alpha\,.\,\mathtt{throw}\,\alpha\,r \to \mathtt{catch}\,\alpha\,.\,r$. The result follows from the induction hypothesis as we have $\nu(r) \leq \nu(\mathtt{throw}\,\alpha\,r)$ and $\ell(r) < \ell(\mathtt{throw}\,\beta\,r)$.

2. Let $\mathtt{catch}\,\alpha\,.\,\mathtt{throw}\,\beta\,v \to \mathtt{throw}\,\beta\,v$. The result holds by Lemma 4.8.

3. Let $\mathtt{catch}\,\alpha\,.\,v \to v$. The result holds by Lemma 4.8. □

Lemma 4.16. *If $r \in [\![\rho]\!]$, $s \in [\![\sigma \to [\sigma] \to [\sigma]]\!]$, and $t \in [\![[\sigma]]\!]$, then* $\mathtt{lrec}\,r\,s\,t \in [\![\rho]\!]$.

Proof. We prove this result by well-founded induction on $\nu(r) + \nu(s) + \nu(t) + \ell_n(t)$. By Lemma 4.12 it is sufficient to show that for each q with $\mathtt{lrec}\,r\,s\,t \to q$ we have $q \in [\![\rho]\!]$. We consider the following interesting reductions.

1. Let $\mathtt{lrec}\,v_r\,v_s\,\mathtt{nil} \to v_r$. The result holds by assumption.

2. Let $\mathtt{lrec}\,v_r\,v_s\,(v_h :: v_t) \to v_s\,v_h\,v_t\,(\mathtt{lrec}\,v_r\,v_s\,v_t)$. By the definition of $v_h :: v_t \in [\![[\sigma]]\!]$ we obtain that $v_h \in [\![\sigma]\!]$ and $v_t \in [\![[\sigma]]\!]$. Therefore, we have $\mathtt{lrec}\,v_r\,v_s\,v_t \in [\![\rho]\!]$ by the induction hypothesis as $\ell_n(v_t) \leq \ell_n(v_h :: v_t)$. Now, the result follows from the assumption.

3. Let $\mathtt{lrec}\,(\mathtt{throw}\,\alpha\,r)\,s\,t \to (\mathtt{throw}\,\alpha\,r)\,s\,t$. By assumption and Lemma 4.8 we have $r, s, t \in \mathtt{SN}$, hence $(\mathtt{throw}\,\alpha\,r)\,s\,t \in [\![\rho]\!]$ by Corollary 4.10. □

Corollary 4.17. *If $x_1 : \rho_1, \ldots, x_n : \rho_n; \Delta \vdash t : \tau$ and $r_i \in [\![\rho_i]\!]$ for all $1 \leq i \leq n$, then*

$$t[x_1 := r_1, \ldots, x_n := r_n] \in [\![\tau]\!].$$

Proof. We prove this result by induction on the derivation of $\Gamma; \Delta \vdash t : \tau$. All cases follow immediately from the results proven in this section. □

Theorem 4.18 (Strong normalization). *If $\Gamma; \Delta \vdash t : \rho$, then $t \in \mathtt{SN}$.*

Proof. We have $x_i \in [\![\rho_i]\!]$ for each $x_i : \rho_i \in \Gamma$ by Lemma 4.8. Therefore, $t \in [\![\rho]\!]$ by Corollary 4.17 and hence $t \in \mathtt{SN}$ by Lemma 4.8. □

5 Conclusions

In this paper we have defined $\lambda::\mathtt{catch}$ and proven that it satisfies the usual meta theoretical properties: subject reduction, progress, confluence, and strong normalization. These proofs require minor extensions of well-known proof methods. This section concludes with some remarks on possible extensions.

An obvious extension is to add more simple data types, like products, sums, finitely branching trees, *etc*. We expect our proofs to extend easily to these data types. However, adding more complex data types presents some challenges. For example, consider the type $\mathtt{tree}$ of unlabeled trees with infinitary

branching nodes, with the constructors $\texttt{leaf} : \texttt{tree}$ and $\texttt{node} : (\texttt{N} \to \texttt{tree}) \to \texttt{tree}$. A naive extension of the $\to$-free restriction would not forbid $\texttt{catch}\,\alpha.\texttt{node}\,(\lambda x.\texttt{throw}\,\alpha\,\texttt{leaf})$ which does not reduce to a value. It would be interesting to modify the $\to$-free restriction to avoid this.

Instead of using a Gödel's **T** style recursor, it would be interesting to consider a system with a pattern match and fixpoint construct. First of all, this approach is more convenient as Gödel's **T** style recursors only allows recursion on direct subterms. Secondly, this approach would avoid the need for tricks as in Example 2.10 to improve efficiency.

Another useful extension is to add second-order types à la System **F**. Doing this in a naive way results in either a loss of subject reduction (if we define type variables to be $\to$-free) or makes using $\texttt{catch}$ and $\texttt{throw}$ for the second-order fragment impossible (if we define type variables not to be $\to$-free).

Instead of using the statically bound control operators $\texttt{catch}$ and $\texttt{throw}$, it would be interesting to consider their dynamically bound variants. In a dynamically bound $\texttt{catch}$ and $\texttt{throw}$ mechanism, that is for example used in the programming language Common Lisp, substitution is not capture avoiding for continuation variables. We do not see problems to use such a mechanism instead.

The further reaching goal of this paper is to define a λ-calculus with data types and control operators that allows program extraction from proofs constructed using classical reasoning. In such a calculus one can write specifications of programs, which can be proven using (a restricted form of) classical logic. Program extraction would then allow to extract a program from such a proof where the classical reasoning steps are extracted to control operators. Herbelin's IQC_{MP}-calculus [Her10] could be interesting as it includes first-order constructs.

This goal is particularly useful for obtaining provably correct algorithms where the use of control operators would really pay off (for example if a lot of backtracking is performed). See [CGU00] for applications to classical search algorithms. The work of Makarov [Mak06] may also be useful here, as it gives ways to optimize program extraction to make it feasible for practical programming.

References

[Bar84]   H. P. Barendregt. *The lambda calculus: its syntax and semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[BHF01]   K. Baba, S. Hirokawa, and K. Fujita. Parallel Reduction in Type Free λ_μ-calculus. *ENTCS*, 42:52–66, 2001. doi:10.1016/S1571-0661(04)80878-8.

[BU02]   G. Barthe and T. Uustalu. CPS Translating Inductive and Coinductive Types. In *PEPM*, pages 131–142. ACM, 2002. doi:10.1145/509799.503043.

[CF98]   L. Colson and D. Fredholm. System T, call-by-value and the minimum problem. *Theoretical Computer Science*, 206(1-2):301 – 315, 1998. doi:10.1016/S0304-3975(98)00011-5.

[CGU00]   J. L. Caldwell, I. P. Gent, and J. Underwood. Search Algorithms in Type Theory. *Theoretical Computer Science*, 232(1-2):55–90, 2000. doi:10.1016/S0304-3975(99)00170-X.

[CP11]   T. Crolard and E. Polonowski. A program logic for higher-order procedural variables and non-local jumps, 2011. Technical report TR-LACL-2011-4. http://arxiv.org/abs/1112.1554.

[Cro99]   T. Crolard. A confluent lambda-calculus with a catch/throw mechanism. *Journal of Functional Programming*, 9(6):625–647, 1999.

[DN05]   R. David and K. Nour. Why the usual candidates of reducibility do not work for the symmetric λ_μ-calculus. *ENTCS*, 140:101–111, 2005. doi:10.1016/j.entcs.2005.06.020.

[GKM12] H. Geuvers, R. Krebbers, and J. McKinna. The $\lambda\mu^{\mathbf{T}}$-calculus. Annals of Pure and Applied Logic, 2012. doi:10.1016/j.apal.2012.05.005.

[Gri90]   T. G. Griffin. A Formulae-as-Types Notion of Control. In *POPL*, pages 47–58. ACM, 1990. doi:10.1145/96709.96714.

[GTL89]   J. Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[Her10]   H. Herbelin. An Intuitionistic Logic that Proves Markov's Principle. In *LICS*, pages 50–56. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.49.

[Mak06]   Y. Makarov. Practical program extraction from classical proofs. In *MFPS*, volume 155 of *ENTCS*, pages 521 – 542, 2006. doi:10.1016/j.entcs.2005.11.071.

[Nak03]   K. Nakazawa. Confluency and Strong Normalizability of Call-by-Value λ_μ-calculus. *Theoretical Computer Science*, 290(1):429–463, 2003. doi: 10.1016/S0304-3975(01)00380-2.

[Par92]   M. Parigot. λ_μ-calculus: An Algorithmic Interpretation of Classical Natural Deduction. In *LPAR*, volume 624 of *LNCS*, pages 190–201, 1992. doi:10.1007/BFb0013061.

[Par93]   M. Parigot. Classical Proofs as Programs. In *Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 263–276, 1993. doi:10.1007/BFb0022575.

[Par97]   M. Parigot. Proofs of Strong Normalisation for Second Order Classical Natural Deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997. doi:10.2307/2275652.

[Py98]   W. Py. *Confluence en λ-calcul*. PhD thesis, Université de Savoie, 1998.

[RS94]   J. Rehof and M. H. Sørensen. The λ_Δ-calculus. In *TACS*, volume 789 of *LNCS*, pages 516–542, 1994. doi:10.1007/3-540-57887-0_113.

[Tai67]   W. W. Tait. Intensional Interpretations of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967. doi:10.2307/2271658.

[Tak95]   M. Takahashi. Parallel Reductions in λ-Calculus. *Information and Computation*, 118(1):120–127, 1995. doi:10.1006/inco.1995.1057.