BRICS

Basic Research in Computer Science

A Temporal-Logic Approach to Binding-Time Analysis

Rowan Davies

See back inner page for a list of recent publications in the BRICS
Report Series. Copies may be obtained by contacting:

BRICS publications are in general accessible through WWW and
anonymous FTP:

```
http://www.brics.dk/
ftp ftp.brics.dk (cd pub/BRICS)
```

A Temporal-Logic Approach

to Binding-Time Analysis

Rowan Davies[*]

(rowan@cs.cmu.edu) [†]

January 15, 1996

Abstract

The Curry-Howard isomorphism identifies proofs with typed λ-calculus terms, and correspondingly identifies propositions with types. We show how this isomorphism can be extended to relate constructive temporal logic with binding-time analysis. In particular, we show how to extend the Curry-Howard isomorphism to include the $\bigcirc$ ("next") operator from linear-time temporal logic. This yields the simply typed $\lambda^{\bigcirc}$-calculus which we prove to be equivalent to a multi-level binding-time analysis like those used in partial evaluation.

Keywords: Curry-Howard isomorphism, partial evaluation, binding-time analysis, temporal logic.

1

1   Introduction

Partial evaluation [8] is a method for specializing a program given part of the program's input. The basic technique is to execute those parts of the program that do not depend on the unknown data, while constructing a residual program from those parts that do. Offline partial evaluation uses a binding-time analysis to determine those parts of the program that will not depend on the unknown (dynamic) data, regardless of the actual value of the known (static) data.

Binding-time analyzes are usually described via typed languages that include binding-time annotations, as for example by Nielson and Nielson [10] and Gomard and Jones [6]. However, the motivation for the particular typing rules that are chosen is often not clear. There has been some work, for example by Palsberg [11], on modular proofs that binding-time analyzes generate annotations that allow large classes of partial evaluators to specialize correctly. However this still does not provide a direct motivation for the particular rules used in binding-time type systems.

In this paper we give a logical construction of a binding-time type system based on temporal logic. Temporal logic is an extension of classical logic to include proofs that formulas are valid at particular times. The Curry-Howard [7] isomorphism relates constructive proofs to typed λ-terms and formulas to types. Thus, we expect that extending the Curry-Howard isomorphism to constructive temporal logic should yield a typed λ-calculus that expresses that a result of a particular type will be available at a particular time. This is exactly what a binding-time type system should capture.

Many different temporal logics and many different temporal operators have been studied, so we need to determine exactly which are relevant to binding-time analysis. In a binding-time separated program, one stage in the program can manipulate as data the code of the following stage. At the level of types this suggests that at each stage we should have a type for code of the next stage. Thus, via the Curry-Howard isomorphism we are led to consider the temporal logic $\bigcirc$ operator, which denotes truth at the next stage, i.e. $\bigcirc A$ is valid if A is valid at the next time. Further, since temporal logics generally allow an unbounded number of "times", they should naturally correspond to a binding-time analysis with many levels, such as that studied by Glück and Jørgensen [5]. The more traditional two-level binding-time analyzes can then be trivially obtained by restriction. Also, in binding-time analysis we have a simple linear ordering of the binding times, so we consider linear-time temporal logic, in which each time has a unique

time immediately following it. Putting this all together naturally suggests that linear-time temporal logic with $\bigcirc$ and multi-level binding-time analysis should be images of each other under the Curry-Howard isomorphism. This does not seem to have been observed previously, and in this paper we show formally that this is indeed the case. The development is relatively straight-forward and our main interest is in demonstrating the direct logical relationship between binding-time analysis and temporal logic.

The structure of this paper is then as follows. In the following section, we start with $L^{\bigcirc}$, an axiomatic formulation due to Stirling [13] for a small classical linear-time temporal logic including $\bigcirc$. We then formulate a natural-deduction system in a similar style to the modal systems of Martini and Masini [9], and prove that it has the same theorems as the axiomatic formulation. This allows us to directly apply the Curry-Howard isomorphism to the natural-deduction system, yielding the typed $\lambda^{\bigcirc}$-calculus with the $\bigcirc$ operator in the types.

In the second half of the paper we consider $\lambda^{\mathbf{m}}$, which is essentially the λ-calculus fragment of the language used in the multi-level binding-time analysis of Glück and Jørgensen [5]. We then construct a simple isomorphism between $\lambda^{\mathbf{m}}$ and $\lambda^{\bigcirc}$ that preserves typing, thus showing that these languages are equivalent as type systems. We conclude by discussing some practical concepts from binding-time analysis.

This work is similar to work by Davies and Pfenning [3] which shows that a typed language Mini-ML$^{\Box}$ based on modal logic captures a powerful form of staged computation, including run-time code generation. They also show that Mini-ML$^{\Box}$ is a conservative extension of the two-level and (linearly ordered) B-level languages studied by Nielson and Nielson [10]. However, they note that this system only allows programs that manipulate closed code, while the binding-time type systems used in partial evaluation, such as that of Gomard and Jones [6], allow manipulation of code with free variables. Thus, the original motivation for the present work was to consider how to extend Mini-ML$^{\Box}$ to a system that is a conservative extension of the binding-time type systems used in partial evaluation. In this paper we achieve that goal, though find that we also lose the features of Mini-ML$^{\Box}$ beyond ordinary binding-time analysis. Our conclusion is that the $\Box$ operator in Mini-ML$^{\Box}$ corresponds to a type for *closed code*, while the $\bigcirc$ operator in $\lambda^{\bigcirc}$ corresponds to a type for *code with free variables*. Thus the two operators are suitable for different purposes, and which one is preferred depends on the context. This suggests that a desirable direction for future work would be the design of a type system correctly capturing both closed code and

3

code with free variables within a single framework.

2 A temporal λ-calculus

In this section we will show how to extend the Curry-Howard isomorphism to include the $\bigcirc$ ("next") operator from temporal logic. Temporal logics are generally formulated axiomatically and in classical style, while a natural-deduction intuitionistic formulation is more appropriate for the Curry-Howard isomorphism. We thus start with a sound and complete axiomatic system given by Stirling [13] for the fragment of classical linear-time temporal logic containing only $\bigcirc$, $\rightarrow$ and $\neg$. Starting from this small fragment allows us to ignore the other connectives while formulating an equivalent natural-deduction system. We then drop $\neg$ and classical reasoning from the natural-deduction system, since our real interest is in the $\bigcirc$ operator, and consider the extension of the Curry-Howard isomorphism by adding proof terms to yield a simply typed λ-calculus with the $\bigcirc$ operator in the types.

2.1 Axiomatic formulation

The following axioms and inference rules for the fragment of classical linear-time temporal logic containing only $\bigcirc$, $\rightarrow$ and $\neg$ are due Stirling [13] (page 516), who shows that they are sound and complete for unravelled models of this logic. We choose this system as our starting point because it appears to be the smallest linear temporal logic containing the $\bigcirc$ operator that has been previously considered in the literature.

$$
\begin{array}{lll}
\text{Axioms:} & \text{L1} & \text{Any classical tautology instance} \\
& \text{L3} & \bigcirc \neg A \;\leftrightarrow\; \neg \bigcirc A \\
& \text{L4} & \bigcirc (A_1 \rightarrow A_2) \;\rightarrow\; (\bigcirc A_1 \rightarrow \bigcirc A_2) \\
\\
\text{Inference rules:} & \text{MP} & \text{if } A_1 \rightarrow A_2 \text{ and } A_1 \text{ then } A_2 \\
& \text{RO} & \text{if } A \text{ then } \bigcirc A
\end{array}
$$

Note that in the inference rules, we require that there be proofs from no assumptions.

4

2.2 Natural-deduction formulation

We now consider a natural-deduction formulation of $L^{\bigcirc}$ and show that it is equivalent to the axiomatic one. We include negation and classical reasoning here so that this equivalence is exact, even though we will drop them later when we consider the typed λ calculus. By presenting the $\bigcirc$ operator using only an introduction rule and elimination rule we have separated it from negation and classical reasoning, allowing us to remove those rules without affecting the basic properties of $\bigcirc$.

Our natural-deduction formulation uses a judgement annotated with a natural number n, representing the "time" of the conclusion and with each assumption A in Γ also annotated by a time n. These are just like the "levels" in the modal natural deduction systems of Martini and Masini [9], and in fact our system is exactly the same as their rules for modal K, except that because of linearity we do not need any restriction on the introduction rule for $\bigcirc$. Our rules for the non-temporal fragment are relatively standard for natural deduction for pure classical logic, which will later allow us to depend on the equivalence between the axiomatic and natural-deduction systems for pure classical logic. We use a sequent style presentation here to correspond with the λ-calculus typing rules presented later.

$$\frac{A^n \text{ in } \Gamma}{\Gamma \vdash^n A} V$$

$$\frac{\Gamma, A_1^n \vdash^n A_2}{\Gamma \vdash^n A_1 \to A_2} \to I \qquad \frac{\Gamma \vdash^n A_1 \to A_2 \qquad \Gamma \vdash^n A_1}{\Gamma \vdash^n A_2} \to E$$

$$\frac{\Gamma, A^n \vdash^n p}{\Gamma \vdash^n \neg A} \neg I^p \quad (p \text{ not in } \Gamma,\ A) \qquad \frac{\Gamma \vdash^n A \qquad \Gamma \vdash^n \neg A}{\Gamma \vdash^n B} \neg E$$

$$\frac{\Gamma, \neg A^n \vdash^n A}{\Gamma \vdash^n A} C$$

$$\frac{\Gamma \vdash^{n+1} A}{\Gamma \vdash^n \bigcirc A} \bigcirc I \qquad \frac{\Gamma \vdash^n \bigcirc A}{\Gamma \vdash^{n+1} A} \bigcirc E$$

In order to give a proof-theoretic semantics to the $\bigcirc$ operator we also

5

need to consider a proof reduction rule for $\bigcirc I$ immediately followed by $\bigcirc E$, which reduces trivially to the proof with both inference steps removed.

The following theorem justifies the above formulation:

Theorem 1 *We can derive* $\cdot \vdash^0 A$ *if and only if there is a proof of A in $L^{\bigcirc}$.*

Proof: (sketch) In one direction, we construct a derivation for each axiom and proceed by induction over the inference rules in $L^{\bigcirc}$. For L1 we actually simply notice that removing the rules for $\bigcirc$ yields a standard natural-deduction system for pure classical logic. The other axioms are straightforward, and the case for the inference rule RO only requires showing that incrementing every time annotation in a derivation yields a derivation.

In the other direction, we prove by induction over the structure of derivations, strengthening the induction hypothesis to:

$$\text{if} \quad A_1^{n_1}, \ldots, A_k^{n_k} \vdash^n A$$
$$\text{then} \quad \bigcirc^{n_1} A_1 \to \ldots \to \bigcirc^{n_k} A_k \to \bigcirc^n A \text{ is provable in } L^{\bigcirc}$$

Here $\bigcirc^n$ means n occurrences of $\bigcirc$. Only the cases for the $\to$ and $\neg$ rules are non-trivial. They are solved by repeated application of L3, L4 and the converse of L4 (which is derivable using L3, L4 and a classical tautology) along with a sequence of cuts (which are classical tautologies) to reduce to the corresponding cases for pure classical logic of the equivalence between natural deduction and axiomatic presentations. $\square$

2.3   A temporal λ-calculus

We now add proof terms to the intuitionistic fragment without $\neg$ of the natural-deduction system. We remove $\neg$ because it is not usually included in typed λ-calculi (though it can be), and in particular it is not included in binding-time type systems. We are justified in simply removing the inference rules for negation and classical reasoning, since the in the natural-deduction formulation the $\bigcirc$ operator is completely captured by its introduction and elimination rules. This yields $\lambda^{\bigcirc}$, a simply typed λ-calculus with the $\bigcirc$ operator in the types, by the natural extension of the Curry-Howard isomorphism.

2.3.1 Syntax

$$
\begin{array}{llll}
\text{Types} & A & ::= & b \quad | \: A_1 \rightarrow A_2 \: | \bigcirc A \\
\text{Terms} & M & ::= & x \quad | \: \lambda x{:}A.\; M \: | \: M_0\, M_1 \\
& & & | \: \mathbf{next}\; M \: | \: \mathbf{prev}\; M \\
\text{Contexts} & \Gamma & ::= & \cdot \quad | \: \Gamma, x{:}A^n
\end{array}
$$

2.3.2 Typing rules

$\Gamma \vdash^n M : A$ Expression M has type A at time n in context Γ.

$$
\frac{x{:}A^n \text{ in } \Gamma}{\Gamma \vdash^n x : A}\; V
$$

$$
\frac{\Gamma, x{:}A_1^n \vdash^n M : A_2}{\Gamma \vdash^n \lambda x{:}A_1.\; M : A_1 \rightarrow A_2}\; {\rightarrow} I
$$

$$
\frac{\Gamma \vdash^n M_0 : A_1 \rightarrow A_2 \qquad \Gamma \vdash^n M_1 : A_1}{\Gamma \vdash^n M_0\, M_1 : A_2}\; {\rightarrow} E
$$

$$
\frac{\Gamma \vdash^{n+1} M : A}{\Gamma \vdash^n \mathbf{next}\; M : \bigcirc A}\; \bigcirc I
\qquad\qquad
\frac{\Gamma \vdash^n M : \bigcirc A}{\Gamma \vdash^{n+1} \mathbf{prev}\; M : A}\; \bigcirc E
$$

2.3.3 Reduction rules

We have the standard β-reduction rule as well as the following reduction rule from the natural-deduction proof-reduction rule, analogously to the correspondence between β-reduction and the proof reduction rule for $\rightarrow$:

$$
\mathbf{prev}(\mathbf{next}\; M) \xrightarrow{\;\bigcirc\;} M
$$

We also have the following for elimination followed by introduction, analogous to η-reduction:

$$
\mathbf{next}(\mathbf{prev}\; M) \xrightarrow{\;\bigcirc\;} M
$$

3 Equivalence to a binding-time type system

We now demonstrate the relationship between $\lambda^\bigcirc$ and binding-time type systems considered by other authors. To do this we consider $\lambda^\mathbf{m}$, a simply typed λ-calculus which is essentially the core of standard binding-time analyzes used in offline partial evaluation (see e.g. Gomard and Jones [6]). $\lambda^\mathbf{m}$ additionally allows more than two binding times in the same way as the multi-level binding-time analysis of Glück and Jørgensen [5]. Our formulation of $\lambda^\mathbf{m}$ is basically the λ-calculus fragment of Glück and Jørgensen's system, though it has some important differences. We use separate syntactic categories for the types of each level, thus avoiding side conditions regarding well-formedness of types. Further, we do not treat the final level as dynamically typed, but consider the whole program to be statically typed. Finally, we do not include "lifting" from one binding time to a later one, but instead demonstrate later in this section how this can be easily added to $\lambda^\mathbf{m}$.

We then give a simple translation between $\lambda^\mathbf{m}$ and $\lambda^\bigcirc$. This translation is a bijection on terms and types that preserves typing, modulo reduction of **prev** - **next** redices. Thus $\lambda^\mathbf{m}$ and $\lambda^\bigcirc$ are equivalent as type systems, modulo these trivial reductions.

3.1 Syntax

We use the separate syntactic categories τ^n to indicate the type of results which will be available at time n or later. The time annotations on base types b^n and function types $\tau_1^n \to^n \tau_2^n$ indicate the time at which the corresponding values are available. The time annotations on terms indicate the time at which the λ or @ (application) are reduced, and the corresponding variable substituted for. See Glück and Jørgensen [5] for a semantics of evaluation of multi-level terms in multiple stages.

$$
\begin{array}{llcll}
\text{Types} & \tau^n & ::= & b^n & \mid \tau_1^n \xrightarrow{n} \tau_2^n \mid \tau^{n+1} \\
\text{Terms} & E & ::= & x^n & \mid \lambda^n x^n{:}\tau^n.\, E \mid E_0 \ @^n\ E_1 \\
\text{Contexts} & \Psi & ::= & \cdot & \mid \Psi, x^n{:}\tau^n
\end{array}
$$

3.2   Typing rules

$$\frac{x^n{:}\tau^n \text{ in } \Psi}{\Psi \vdash^{\mathbf{m}} x^n : \tau^n} \text{ tpm\_var} \qquad\qquad \frac{\Psi, x^n{:}\tau_1^n \vdash^{\mathbf{m}} E : \tau_2^n}{\Psi \vdash^{\mathbf{m}} \lambda^n x^n{:}\tau^n.\ E : \tau_1^n \xrightarrow{n} \tau_2^n} \text{ tpm\_lam}$$

$$\frac{\Psi \vdash^{\mathbf{m}} E_0 : \tau_1^n \xrightarrow{n} \tau_2^n \qquad \Psi \vdash^{\mathbf{m}} E_1 : \tau_1^n}{\Psi \vdash^{\mathbf{m}} E_0 \ @^n\ E_1 : \tau_2^n} \text{ tpm\_app}$$

3.3   Equivalence translation

We now give simple translations between well typed terms in $\lambda^{\mathbf{m}}$ and $\lambda^{\bigcirc}$. The translation from $\lambda^{\bigcirc}$ maps terms which are in the same equivalence class with respect to $\mathbf{next}(\mathbf{prev}\ M)$ and $\mathbf{prev}(\mathbf{next}\ M)$ reductions to the same $\lambda^{\mathbf{m}}$ term. In the other direction, we always translate $\lambda^{\mathbf{m}}$ terms to $\lambda^{\bigcirc}$ terms with all such redices reduced, these being unique representatives of the equivalence classes. Note that we can always reduce all these redices, since the number of reductions is bounded by the number of $\mathbf{next}$ and $\mathbf{prev}$ constructors. We then show that the two translations are inverses of each other when restricted to the representatives of the equivalence classes, and that they preserve typing. This shows that $\lambda^{\mathbf{m}}$ is isomorphic to $\lambda^{\bigcirc}$ modulo the $\mathbf{next}$-$\mathbf{prev}$ reductions. Thus they are equivalent, modulo these trivial reductions.

The translations are given by the functions $|\cdot|^n$ and $\|\cdot\|^n$ defined as follows:

Type Translation

$$
\begin{array}{rclcrcl}
|b^n|^n & = & b & & \|b\|^n & = & b^n \\
|\tau_1^n \rightarrow^n \tau_2^n|^n & = & |\tau_1^n|^n \rightarrow |\tau_2^n|^n & & \|A_1 \rightarrow A_2\|^n & = & \|A_1\|^n \rightarrow^n \|A_2\|^n \\
|\tau^{n+1}|^n & = & \bigcirc|\tau^{n+1}|^{n+1} & & \|\bigcirc A\|^n & = & \|A\|^{n+1}
\end{array}
$$

Term Translation

$$
\begin{array}{rclcrcl}
|x^n|^n & = & x & & \|x\|^n & = & x^n \\
|\lambda^n x^n{:}\tau^n.\ E|^n & = & \lambda x{:}|\tau^n|^n.\ |E|^n & & \|\lambda x{:}A.\ M\|^n & = & \lambda x^n{:}\|A^n\|^n.\ \|M\|^n \\
|E_0 @^n E_1|^n & = & |E_0|^n\ |E_1|^n & & \|M_0\ M_1\|^n & = & \|M_0\|^n @^n \|M_1\|^n \\
(m > n)\quad |E^m|^n & = & \mathbf{next}|E^m|^{n+1} & & \|\mathbf{next}\ M\|^n & = & \|M\|^{n+1} \\
(m < n)\ |E^m|^{n+1} & = & \mathbf{prev}|E^m|^n & & \|\mathbf{prev}\ M\|^{n+1} & = & \|M\|^n
\end{array}
$$

Here we use E^m as convenient syntax matching all expressions which have the top constructor annotated with m.

Lemma 2 $|\cdot|^n$ and $\|\cdot\|^n$ are inverses on the fragment of $\lambda^\bigcirc$ with no **next-prev** redices.

Proof: By a straight-forward induction. □

Theorem 3 The two translations preserve typing, namely:

- if $\cdot \vdash^{\mathbf{m}} E : \tau^0$ then $\cdot \vdash^0 |M|^0 : |\tau^0|^0$

- if $\cdot \vdash^0 M : A$ then $\cdot \vdash^{\mathbf{m}} \|M\|^0 : \|A\|^0$

Proof: By a straight-forward structural induction, strengthening appropriately to:

- if $\Psi \vdash^{\mathbf{m}} E : \tau^n$ then $\Psi \vdash^n |M|^n : |\tau|^n$
- if $\Gamma \vdash^n M : A$ then $\|\Gamma\| \vdash^{\mathbf{m}} \|M\|^n : \|A\|^n$

□

3.4 Relationship to binding-time analysis in practice

The proof above shows that $\lambda^\bigcirc$ and $\lambda^{\mathbf{m}}$ are equivalent as type systems, thus justifying our claim that binding-time type systems are the image of temporal logic under the the Curry-Howard isomorphism. However, $\lambda^{\mathbf{m}}$ is lacks some aspects of real binding-time type systems, and in particular we now compare $\lambda^{\mathbf{m}}$ in detail with the multi-level binding-time type system of Glück and Jørgensen [5].

Firstly, we have verified that there is no difficulty in extending $\lambda^{\mathbf{m}}$ and $\lambda^\bigcirc$ with data-structures, fixed-points, and other features from realistic functional languages, just as for the simply-typed λ-calculus.

Secondly, the "lift" coercions which map values from one binding time to a later one are missing from our language. Glück and Jørgensen [5] include lift coercions only at base types. Since $\lambda^\bigcirc$ and $\lambda^{\mathbf{m}}$ do not include particular

base types and primitive operations, it is not surprising that they do not include lift coercions. Further, we refer to work by Davies and Pfenning [3] in which it was shown that these lift coercions are generally definable as functions in a staged version of a language like ML. For example, in a language extending Standard ML with Next, Prev and 0 from $\lambda^{\bigcirc}$, and a type of natural numbers represented using zero and successor similar to Mini-ML [2], we have the following lift coercion:

```
(* val liftNat : nat -> 0 nat *)
fun liftNat Z      = Next Z
  | liftNat (S x) = Next (S (Prev (liftNat x)))
```

With these lift coercions defined appropriately, $\lambda^{\mathbf{m}}$ is almost identical to the λ-calculus core of the multi-level binding-time analysis of Glück and Jørgensen [5], except for some minor syntactic differences. The only other remaining difference is that Glück and Jørgensen allow the final binding time to be dynamically typed. Since the Curry-Howard isomorphism only concerns statically typed languages, it is not surprising that $\lambda^{\bigcirc}$ and $\lambda^{\mathbf{m}}$ differ in this regard.

Finally, we observe that $\lambda^{\bigcirc}$ allows an interesting perspective on the relationship between the binding-time annotated language and the original language. If we consider **next**, **prev** and the lift coercions to all be implicit coercions in the original language, then we have a type system including $\bigcirc$ for this language that is reminiscent of sub-typing in the style of work by Breazu-Tannen *et.al.* [1]. Doing type inference and making these "coercions" explicit is then exactly a binding-time analysis. However, **next** and **prev** are not functions, so they are quite different from other implicit coercions.

$\lambda^{\bigcirc}$ could also serve as the basis for an extension of a statically typed language like Standard ML to allow hand-written programs that generate specialized code. Welinder [14] has shown this to be a useful technique in some situations, since it gives the programmer direct control over binding-time decisions, and forces them to think about binding-times when writing code. This of course comes at the price that the programmer must manually annotate the program themselves. We consider $\lambda^{\bigcirc}$ more appropriate as the basis for such a language than $\lambda^{\mathbf{m}}$ because it has a less intrusive syntax.

4 Conclusion

We have demonstrated that the image of a small temporal logic under the Curry-Howard isomorphism, $\lambda^\bigcirc$, provides a logical construction of a binding-time type system that is equivalent to those used in partial evaluation. In particular, $\lambda^\bigcirc$ allows programs that manipulate code with free variables. This is in contrast to work by Davies and Pfenning [3] on Mini-ML$^\square$, a typed language based on modal logic that also expresses a form of binding-time analysis. As an example, we have a the following term of type $(A^1 \rightarrow^0 B^1) \rightarrow^0 (A^1 \rightarrow^1 B^1)$ in the binding-time type system $\lambda^{\mathbf{m}}$:

$$\lambda^0 f^0{:}A^1 \rightarrow^0 B^1. \ \lambda^1 x^1{:}A^1. \ f^0 \ @^0 \ x^1$$

There is no corresponding typed term in Mini-ML$^\square$. In $\lambda^\bigcirc$ we can use the translation in the previous section to yield a corresponding term of type $(\bigcirc A \rightarrow \bigcirc B) \rightarrow \bigcirc(A \rightarrow B)$, as follows:

$$\lambda f{:} \bigcirc A \rightarrow \bigcirc B. \ \mathbf{next} \ (\lambda x{:}A. \ \mathbf{prev} \ (f \ (\mathbf{next} \ x)))$$

However, the manipulation of code with free variables comes at a price. Since $\lambda^\bigcirc$ does not express closed code, it can not be directly extended with a construct like that in Mini-ML$^\square$ that expresses evaluation of generated code. Such a construct is essential in a language that supports general forms of staged computation, and is the main novel feature of Mini-ML$^\square$, so in future work we will consider how to construct a type system that captures both closed code and code with free variables.

One possible direction for this work is based on the observation that manipulation of code with free variables is allowed in $\lambda^\bigcirc$ because there is only a single successor stage from any stage, which corresponds to the fact that $\lambda^\bigcirc$ is based on a linear-time temporal logic. In Mini-ML$^\square$ we allow each stage to have several successor stages in order to allow more general forms of staged computation, in particular run-time code generation and sharing of code between stages (see [3] for details). This means that when constructing code in an arbitrary successor stage we cannot use variables that are bound further out in a possibly different successor stage.

This suggests that to design a language which expresses both closed code and code with free variables we could explicitly name stages and provide an explicit quantifier over them, rather than using **next** and **prev** to move between stages. This is similar to the systems of labelled natural deduction

of Gabbay and de Queiroz [4], which allow many different logics to be formulated including modal logics, though this is still a speculative direction for future research.

We have implemented type checkers for the languages $\lambda^\bigcirc$ and $\lambda^{\mathbf{m}}$ in the logic programming language Elf (see Pfenning [12]). Using logic programming variables, the same programs will also perform type inference. We have also implemented the translations and proof of equivalence between these languages in Elf.

5 Acknowledgements

References

[1] Val Breazu-Tannen, Thierry Coquand, Carl Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991.

[2] Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *Proceedings of the 1986 Conference on LISP and Functional Programming*, pages 13–27. ACM Press, 1986.

[3] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Programming Languages*, January 1996. To appear. Earlier available as Technical Report CMU-CS-95-145, Carnegie Mellon University, May 1995.

[4] Dov M. Gabbay and Ruy J.G.B. de Queiroz. Extending the curry-howard interpretation to linear, relevant and other resource logics. *Journal of Symbolic Logic*, 57:1319–1365, 1992.

[5] Robert Glück and Jesper Jørgensen. Efficient multi-level generating extensions for program specialization. In S.D. Swierstra and M. Hermenegildo, editors, *Programming Languages, Implementations, Logics and Programs (PLILP'95)*, volume 982 of *Lecture Notes in Computer Science*, pages 259–278. Springer-Verlag, September 1995.

[6] Carsten Gomard and Neil Jones. A partial evaluator for the untyped lambda-calculus. *Journal of Functional Programming*, 1(1):21–69, January 1991.

[7] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, 1980*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard, 1979.

[8] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International Series in Computer Science. Prentice-Hall, 1993.

[9] Simone Martini and Andrea Masini. A computational interpretation of modal proofs. In H. Wansing, editor, *Proof Theory of Modal Logics*. Kluwer, 1995. To appear.

[10] Flemming Nielson and Hanne Riis Nielson. *Two-Level Functional Languages*. Cambridge University Press, 1992.

[11] Jens Palsberg. Correctness of binding time analysis. *Journal of Functional Programming*, 3(3):347–363, July 1993.

[12] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[13] Colin Stirling. Modal and temporal logics. In Samson Abramsky, Dov M. Gabby, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 2*, chapter 5, pages 477–563. Oxford University Press, Oxford, 1992.

[14] Morten Welinder. Very efficient conversions. In E. Thomas Schubert, Phillip J. Windley, and James Alves-Foss, editors, *The 8th International*

Workshop on Higher Order Logic Theorem Proving and Its Applications, Aspen Grove, Utah, volume 971 of *Lecture Notes in Computer Science*, pages 340–352. Springer Verlag, September 1995.

Recent Publications in the BRICS Report Series

RS-95-51 Rowan Davies. *A Temporal-Logic Approach to Binding-Time Analysis*. October 1995. 15 pp.

RS-95-50 Dany Breslauer. *On Competitive On-Line Paging with Lookahead*. September 1995. 12 pp.

RS-95-49 Mayer Goldberg. *Solving Equations in the λ-Calculus using Syntactic Encapsulation*. September 1995. 13 pp.

RS-95-48 Devdatt P. Dubhashi. *Simple Proofs of Occupancy Tail Bounds*. September 1995. 7 pp. To appear in *Random Structures and Algorithms*.

RS-95-47 Dany Breslauer. *The Suffix Tree of a Tree and Minimizing Sequential Transducers*. September 1995. 15 pp.

RS-95-46 Dany Breslauer, Livio Colussi, and Laura Toniolo. *On the Comparison Complexity of the String Prefix-Matching Problem*. August 1995. 39 pp. Appears in Leeuwen, editor, *Algorithms - ESA '94: Second Annual European Symposium proceedings*, LNCS 855, 1994, pages 483–494.

RS-95-45 Gudmund Skovbjerg Frandsen and Sven Skyum. *Dynamic Maintenance of Majority Information in Constant Time per Update*. August 1995. 9 pp.

RS-95-44 Bruno Courcelle and Igor Walukiewicz. *Monadic Second-Order Logic, Graphs and Unfoldings of Transition Systems*. August 1995. 39 pp. To be presented at CSL '95.

RS-95-43 Noam Nisan and Avi Wigderson. *Lower Bounds on Arithmetic Circuits via Partial Derivatives (Preliminary Version)*. August 1995. 17 pp. To appear in *36th Annual Conference on Foundations of Computer Science*, FOCS '95, IEEE, 1995.

RS-95-42 Mayer Goldberg. *An Adequate Left-Associated Binary Numeral System in the λ-Calculus*. August 1995. 16 pp.

RS-95-41 Olivier Danvy, Karoline Malmkjær, and Jens Palsberg. *Eta-Expansion Does The Trick*. August 1995. 23 pp.