ELSEVIER

Science of Computer Programming 31 (1998) 113–145

Science of Computer Programming

Strictness and totality analysis

Kirsten Lackner Solberg Gasser*, Hanne Riis Nielson, Flemming Nielson

Computer Science Department, Aarhus University, Ny Munkegade, Bldg 540, DK-8000, Aarhus, Denmark

Abstract

We define a novel inference system for strictness and totality analysis for the simply-typed lazy lambda-calculus with constants and fixpoints. Strictness information identifies those terms that definitely denote bottom (i.e. do not evaluate to WHNF) whereas totality information identifies those terms that definitely do not denote bottom (i.e. do evaluate to WHNF). The analysis is presented as an annotated type system allowing conjunctions at "top-level" only. We give examples of its use and prove the correctness with respect to a natural-style operational semantics.

Keywords: Strictness and totality analysis; Top-level conjunction types; Natural-style operational semantics; Annotated type system

1. Introduction

Strictness analysis has proved useful in the implementation of lazy functional languages like Miranda, Lazy ML and Haskell: when a function is strict it is safe to evaluate its argument before performing the function call. In the literature there are several approaches to the specification of strictness analysis: abstract interpretation (e.g. [12, 3]), projection analysis (e.g. [21]), and inference based methods (e.g. [2, 6, 8, 9, 22]).

Totality analysis is in a sense dual to strictness analysis: if the argument to a function is known to terminate then it is safe to evaluate it before performing the function call [11]. Totality analysis has received much less attention than strictness analysis and has primarily been specified using abstract interpretation [12, 1]. In a sense it can be regarded as an approximation to time complexity analysis.

In this paper we present an inference system for performing combined strictness and totality analysis. Three annotations on underlying types, ut, are introduced:
– $\mathtt{ut}^{\mathbf{b}}$: the value has type ut and is definitely $\perp$,

* Corresponding author. E-mail: {kls, hrn, fn}@daimi.aau.dk.

0167-6423/98/$19.00
PII S0167-6423(96)00043-3

– $\mathtt{ut}^{\mathbf{n}}$: the value has type ut and is definitely *not* $\bot$, and
– $\mathtt{ut}^{\top}$: the value has type ut and can be any value.

Annotated types can be constructed using the function type constructor and top-level conjunction. As an example a function may have the annotated type $(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$ which means that given a terminating argument the function will definitely terminate and given a non-terminating argument it will definitely not terminate. Thus we capture the strictness as well as the totality of the function. Strictness and totality information can also be combined as in

$$(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

which will be the annotated type of McCarthy's ambiguity operator: if one of the two arguments terminate so does the function call but if both arguments diverge so does the function call.

We shall claim that the inference based approach allows a natural combination of the two analyses. In contrast Mycroft [12] presents both analyses using abstract interpretation but as separate analyses: the strictness analysis is based on downwards closed sets and the totality analysis on upwards closed sets. While we believe that the two analyses could be combined using the convex power domain of [13] we find this untractable because establishing the mathematical foundations will be a rather formidable task and extensions to richer languages would not be easy. A more recent approach to use abstract interpretation to combine the two analyses is the comportment analysis of [4].

The semantic foundations of our work is based on natural style operational semantics [5, 16]. We employ a lazy semantics so that terms are evaluated to weak head normal form (WHNF). Consequently we capture the semantics of "real-life" lazy functional languages in contrast to most other papers on strictness analysis (like [3]) where terms are evaluated to head normal form.

1.1. Motivating example

Example 1. Consider the CBN program

```
let f = λg . λx . g (x)
    a = ...
in  f (λx . x) a
```

A naive CBV version of it may be

```
let f = λg . λx . (g ()) (x)
    a = λ() . ...
in  f (λ() . λx . x ()) a
```

However, an optimised CBV version is:

```
let f = λg . λx . g(x)
    a = ...
in  f (λx.x) a
```

since f is strict in its first argument we need not thunkify the first argument to f and since the first argument to f is always a strict function we need not thunkify the second argument to f. This can be seen from the strictness type of f:

$$((\mathtt{Int} \to \mathtt{Int})^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}) \wedge ((\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}) \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

Now consider the CBN program

```
let f = λg . λx . g (x)
    h = ...
in  f h 1
```

A naive CBV version of it may be

```
let f = λg . λx . (g ()) (x)
    h = λ() . ...
in  f h (λ() . 1)
```

However, an optimised CBV version is:

```
let f = λg . λx . g (x)
    h = ...
in  f h 1
```

since, again, f is strict in its first argument we need not thunkify the first argument to f and since the argument to g (i.e. the second argument to f) is terminating we need not thunkify it. This information can be gained from the strictness type of f:

$$(\mathtt{Int} \to \mathtt{Int})^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}$$

and the totality type of f:

$$(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}$$

Now let us combine the two examples into one:

```
let f = λg . λx . g (x)
    a = ...
    h = ...
in  f (λx . x) a + f h 1
```

A naive CBV version of it may be

```
let f = λg . λx . (g ()) (x)
    a = λ() . ...
    h = λ() . ...
in  f (λ() . λx . x ()) a + f h (λ() . 1)
```

The strictness type of f is

$$((\mathtt{Int} \to \mathtt{Int})^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}) \wedge ((\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}) \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

However, we cannot remove the thunkification of the second argument to f since in the second call to f the first argument is not a strict function. So what we get is

```
let f = λg . λx . g (x)
    a = λ() . ...
    h = ...
in  f (λx . x) a + f h (λ() . 1)
```

The totality type of f is:

$$(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}$$

We cannot use this information to remove the thunkification of the second argument to f since in the first call to f the second argument need not terminate.
But from the strictness and totality type of f:

$$((\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge ((\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}}) \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

we can indeed remove the thunkification of the second argument to f.
This example shows clearly that we get more information by doing strictness and totality analysis at the same time, instead of do first strictness analysis and then totality analysis. □

1.2. Overview

In Section 2 we define the strictness and totality types and give rules for coercing between them; a notion of conjunction type is defined but only at "top-level"; finally the inference system is presented and examples of its use are given. In Section 3 we then present a natural style operational semantics. Finally in Section 4 the analysis is proven semantically sound.

2. The annotated type system

First we present the language and underlying types, then we introduce the annotated types. Using top-level conjunctions of the annotated types we define a strictness and totality analysis.

2.1. The standard type system

We consider the simply typed λ-calculus with constants. The standard types are either base-types (denoted B and including Int and Bool) or function types:

$$\mathtt{ut} ::= \mathtt{B} \mid \mathtt{ut} \to \mathtt{ut}$$

$$[\text{var}]\ \frac{}{A \vdash x : ut}\quad \text{if } (x : ut) \in A$$

$$[\text{abs}]\ \frac{A, x : ut_1 \vdash e : ut_2}{A \vdash \lambda x.e : ut_1 \to ut_2}$$

$$[\text{app}]\ \frac{A \vdash e_0 : ut_1 \to ut_2 \quad A \vdash e_1 : ut_1}{A \vdash e_0\ e_1 : ut_2}$$

$$[\text{if}]\ \frac{A \vdash e_1 : \texttt{Bool} \quad A \vdash e_2 : ut \quad A \vdash e_3 : ut}{A \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : ut}$$

$$[\text{fix}]\ \frac{A \vdash e : ut \to ut}{A \vdash \texttt{fix } e : ut}$$

$$[\text{const}]\ \frac{}{A \vdash c : ut_c}$$

Fig. 1. Type inference.

The terms are given by

$$e ::= x \mid \lambda x.e \mid e\ e \mid c \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid \texttt{fix } e$$

The constants (the c's) include `true` and `false` of type `Bool` and all the integers of type `Int` in addition to +, *, ... of type $\texttt{Int} \to \texttt{Int} \to \texttt{Int}$. We will only consider terms that are typable according to the type inference rules defined in Fig. 1 and for simplicity we shall require that all bound variables be distinct. The list A of assumptions gives types to free variables and each variable only occurs once in the assumption list. For each constant c its type is given by ut_c (e.g. $ut_+ = \texttt{Int} \to \texttt{Int} \to \texttt{Int}$). The set of free variables in the term e is written FV(e) and substitution on terms is written $e[e_2/x]$, where $e[e_2/x]$ is the term e where all free occurrences of x are replaced by e_2.

The meaning of a type judgement, $A \vdash e : ut$, is that under the assumptions, A, for the free variables, the term, e, has the type ut.

Example 2. With the rules in Fig. 1 we can infer

$$\emptyset \vdash \lambda x.x : \texttt{Int} \to \texttt{Int}$$

and $\emptyset \vdash \lambda x.x : \texttt{Real} \to \texttt{Real}$. However we are not able to infer

$$\emptyset \vdash \texttt{if e then 7 else 7.2} : \texttt{Real}$$

to do so we would need to coerce `Int` to `Real` and this is not supported by the type system of Fig. 1.

2.2. Strictness and totality types

A strictness and totality type, t, is either an annotated underlying type or a function type between strictness and totality types:

$$t ::= ut^s \mid t \to t$$

$$ut ::= B \mid ut \to ut$$

$$s ::= \top \mid \mathbf{n} \mid \mathbf{b}$$

The annotations (the s's) can either be $\top$, $\mathbf{n}$, or $\mathbf{b}$. The idea is that a term with the strictness and totality type $\mathtt{ut}^{\mathbf{b}}$ has the underlying type $\mathtt{ut}$ and *does not* evaluate to a WHNF. A term with the strictness and totality type $\mathtt{ut}^{\mathbf{n}}$ has the underlying type $\mathtt{ut}$ and *does* evaluate to a WHNF. Finally a term with the strictness and totality type $\mathtt{ut}^{\top}$ has the underlying type $\mathtt{ut}$ but we do not know anything about the evaluation of the term. A term with the strictness and totality type $\mathtt{t}_1 \to \mathtt{t}_2$ will, when applied to a term with strictness and totality type $\mathtt{t}_1$, yield a term with strictness and totality type $\mathtt{t}_2$.

Example 3. All functions with the underlying type $\mathtt{ut}_1 \to \mathtt{ut}_2$ will also have the strictness and totality types $(\mathtt{ut}_1 \to \mathtt{ut}_2)^{\top}$ and $(\mathtt{ut}_1^{\top} \to \mathtt{ut}_2^{\top})$. A function with no WHNF has the strictness and totality type $(\mathtt{ut}_1 \to \mathtt{ut}_2)^{\mathbf{b}}$ and the function that applied to any term yields a term with no WHNF has the strictness and totality type $\mathtt{ut}_1^{\top} \to \mathtt{ut}_2^{\mathbf{b}}$.

For later reference we define the predicate BOT($\mathtt{t}$) by

$$\mathrm{BOT}(\mathtt{ut}^{\mathbf{b}}) = \mathtt{tt}, \quad \mathrm{BOT}(\mathtt{ut}^{\top}) = \mathtt{tt},$$
$$\mathrm{BOT}(\mathtt{ut}^{\mathbf{n}}) = \mathtt{ff}, \quad \mathrm{BOT}(\mathtt{t}_1 \to \mathtt{t}_2) = \mathrm{BOT}(\mathtt{t}_2)$$

and we shall see that if $\mathrm{BOT}(\mathtt{t}) = \mathtt{tt}$ then there is a term of type $\mathtt{t}$ without any WHNF.

Given an annotated type $\mathtt{t}$ we can speak about the *underlying type* of the type $\mathtt{t}$ as the type obtained by removing all the annotations from the type; we will write $\varepsilon(\mathtt{t})$ for the underlying type of $\mathtt{t}$.

2.2.1. *Coercions between strictness and totality types*

Most terms have more than one strictness and totality type; as an example the strictness and totality types of $\lambda x.7$ include $(\mathtt{Int} \to \mathtt{Int})^{\top}$, $(\mathtt{Int} \to \mathtt{Int})^{\mathbf{n}}$, and $\mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$. Some of these are redundant and to express this we define coercions between them: $\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2$ is to hold if all terms of strictness and totality type $\mathtt{t}_1$ also have strictness and totality type $\mathtt{t}_2$ (and in particular the underlying types are the same).

The coercion relation $\leqslant_{\mathrm{ST}}$ is defined by the rules of Fig. 2. It is reflexive, transitive, and anti-monotonic in contravariant position, and we write $\equiv$ for the equivalence induced by $\leqslant_{\mathrm{ST}}$, i.e. $\mathtt{t}_1 \equiv \mathtt{t}_2$ if and only if $\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2$ and $\mathtt{t}_2 \leqslant_{\mathrm{ST}} \mathtt{t}_1$. The rule [top1] expresses that the strictness and totality type $\mathtt{ut}^{\top}$ is the greatest among the strictness and totality types with the underlying type $\mathtt{ut}$. One axiom derived from the rule [top1] is

$$\mathtt{ut}_1^{\top} \to \mathtt{ut}_2^{\top} \leqslant_{\mathrm{ST}} (\mathtt{ut}_1 \to \mathtt{ut}_2)^{\top} \tag{1}$$

$$[\text{ref}]\ \frac{}{t \leqslant_{ST} t}$$

$$[\text{trans}]\ \frac{t_1 \leqslant_{ST} t_2 \quad t_2 \leqslant_{ST} t_3}{t_1 \leqslant_{ST} t_3}$$

$$[\text{arrow}]\ \frac{t_3 \leqslant_{ST} t_1 \quad t_2 \leqslant_{ST} t_4}{t_1 \to t_2 \leqslant_{ST} t_3 \to t_4}$$

$$[\text{top1}]\ \frac{}{t \leqslant_{ST} \varepsilon(t)^{\top}}$$

$$[\text{top2}]\ \frac{}{(ut_1 \to ut_2)^{\top} \leqslant_{ST} ut_1^{\top} \to ut_2^{\top}}$$

$$[\text{bot}]\ \frac{}{(ut_1 \to ut_2)^{\mathbf{b}} \leqslant_{ST} ut_1^{\top} \to ut_2^{\mathbf{b}}}$$

$$[\text{notbot}]\ \frac{}{ut_1^{\mathbf{n}} \to ut_2^{\mathbf{n}} \leqslant_{ST} (ut_1 \to ut_2)^{\mathbf{n}}}$$

$$[\text{monotone}]\ \frac{}{t_1 \to t_2 \leqslant_{ST} t_1' \to t_2'} \quad \text{if } t_1' = \downarrow t_1 \text{ and } t_2' = \downarrow t_2$$

Fig. 2. Coercions between strictness and totality types.

Axiom (1) then motivates rule [top2] because when combined they yield

$$(ut_1 \to ut_2)^{\top} \equiv ut_1^{\top} \to ut_2^{\top}.$$

The left-hand side of the rule [bot] represents the functions without WHNF and the right-hand side represents all non-terminating functions; this also includes the functions without WHNF. The rule [notbot] says that functions that map terms with a WHNF to a term with WHNF are also included in the functions with a WHNF.

The rule [monotone] ensures that we live in a universe of monotone functions: if we know less about the argument to a function, then we should know less about the result as well. The formulation of this requires the function $\downarrow$ on strictness and totality types defined by

$$\begin{aligned}
\downarrow(ut^{\mathbf{b}}) &= ut^{\mathbf{b}} \\
\downarrow(ut^{\top}) &= ut^{\top} \\
\downarrow(ut^{\mathbf{n}}) &= ut^{\top} \\
\downarrow(t_1 \to t_2) &= t_1 \to \downarrow t_2
\end{aligned}$$

The idea behind $\downarrow$ is that $\downarrow t$ is the "smallest type" (in the sense of "containing" fewest elements) such that both $t \leqslant_{ST} \downarrow t$ and BOT($\downarrow t$) hold; this is formalised in Fact 29 in Section 4.1.

Example 4. To see that the rule [monotone] is useful consider the term `twice` defined by

$$\lambda f.\lambda x.f\ (f\ x)$$

and the strictness and totality type

$$(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}}) \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$

In order to show that `twice` does indeed have that type we must be able to coerce

$$\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \leqslant_{\mathrm{ST}} \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$

However we cannot do so without the [monotone]-rule. For more details see Example 9 below.

We shall later show that the relation $\leqslant_{\mathrm{ST}}$ is sound (Lemma 41). However it is not complete. To see this consider the two strictness and totality types $\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}}$ and $\mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$. It must be the case that every term with the first type also has the second type and vice versa since the terms are monotonic. However, although we can infer

$$\mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}} \leqslant_{\mathrm{ST}} \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}}$$

it turns out that we cannot infer

$$\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}} \leqslant_{\mathrm{ST}} \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$$

using the coercions of Fig. 2. This can be remedied by introducing the rule [monotone2] below: first we define the function $\uparrow$ on strictness and totality types as follows:

$$\uparrow(\mathtt{ut}^{\mathbf{b}}) = \mathtt{ut}^{\top}, \quad \uparrow(\mathtt{ut}^{\top}) = \mathtt{ut}^{\top},$$
$$\uparrow(\mathtt{ut}^{\mathbf{n}}) = \mathtt{ut}^{\mathbf{n}}, \quad \uparrow(\mathtt{t}_1 \to \mathtt{t}_2) = \mathtt{t}_1 \to \uparrow\mathtt{t}_2.$$

The idea behind $\uparrow$ is that it is the "smallest type" such that both $\mathtt{t} \leqslant_{\mathrm{ST}} \uparrow\mathtt{t}$ and

$$\mathrm{NOTBOT}(\uparrow\mathtt{t})$$

hold where the predicate NOTBOT(t) must hold whenever the strictness and totality type must incorporate a term with a WHNF. Now we can write the new coercion rule using $\uparrow$:

$$\text{[monotone2]}\ \frac{}{\mathtt{t}_1 \to \mathtt{t}_2 \leqslant_{\mathrm{ST}} \mathtt{t}_1' \to \mathtt{t}_2'} \quad \text{if } \mathtt{t}_1' = \uparrow\mathtt{t}_1 \text{ and } \mathtt{t}_2' = \uparrow\mathtt{t}_2$$

With this rule we can infer that $\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}} \leqslant_{\mathrm{ST}} \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$. More work is needed to clarify if $\leqslant_{\mathrm{ST}}$ is complete with the new rule added.

Example 5. To see that the rule [monotone2] is useful consider the term `twice` defined by

$$\lambda\mathtt{f}.\lambda\mathtt{x}.\mathtt{f}\ (\mathtt{f}\ \mathtt{x})$$

and the strictness and totality type

$$(\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}}) \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$$

In order to show that `twice` does indeed have that type we must be able to coerce

$$\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}} \leqslant_{\mathrm{ST}} \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{n}}$$

However we cannot do so without the [monotone2]-rule. The details are analogous to Example 9 below.

While we conjecture that adding [monotone2] will be semantically sound the technical machinery needed for characterising the new auxiliary concepts, $\uparrow$ and NOTBOT (corresponding to $\downarrow$ and BOT for [monotone]) in order to formally prove our conjecture is sufficiently involved that we shall dispense with so doing.

2.3. Conjunction types

Based on the strictness and totality types we now define the conjunction types. A conjunction type, `ct`, is either a strictness and totality type or a conjunction of two conjunction types:

$$\begin{aligned}
\mathtt{ct} &::= \mathtt{t} \mid \mathtt{ct} \wedge \mathtt{ct} \\
\mathtt{t} &::= \mathtt{ut}^{s} \mid \mathtt{t} \to \mathtt{t} \\
\mathtt{ut} &::= \mathtt{B} \mid \mathtt{ut} \to \mathtt{ut} \\
s &::= \top \mid \mathbf{n} \mid \mathbf{b}
\end{aligned}$$

Thus conjunction is only allowed at the top-level (just like type-schemes in ML are only allowed at the top-level [10]). The introduction of conjunction types means that it is possible to have empty types like $\mathtt{Int}^{\mathbf{n}} \wedge \mathtt{Int}^{\mathbf{b}}$. Actually, the fine details of empty types are closely connected with the choice of semantic model: emptiness of the type

$$(\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

depends on whether the semantic model allows non-sequential behaviours of type

$$\mathtt{Int} \to \mathtt{Int} \to \mathtt{Int}$$

This will normally be the case for denotational semantics but will not be the case for natural-style operational semantics if the order of evaluation is forced. The restriction to top-level conjunctions allows us to avoid some of the problems introduced by empty types; we return to this later.

$$[\text{ref}]\ \frac{}{\mathtt{ct} \leqslant_{\mathrm{CT}} \mathtt{ct}}$$

$$[\text{trans}]\ \frac{\mathtt{ct}_1 \leqslant_{\mathrm{CT}} \mathtt{ct}_2 \quad \mathtt{ct}_2 \leqslant_{\mathrm{CT}} \mathtt{ct}_3}{\mathtt{ct}_1 \leqslant_{\mathrm{CT}} \mathtt{ct}_3}$$

$$[\wedge 1]\ \frac{}{\mathtt{ct}_1 \wedge \mathtt{ct}_2 \leqslant_{\mathrm{CT}} \mathtt{ct}_1}$$

$$[\wedge 2]\ \frac{}{\mathtt{ct}_1 \wedge \mathtt{ct}_2 \leqslant_{\mathrm{CT}} \mathtt{ct}_2}$$

$$[\wedge 3]\ \frac{\mathtt{ct} \leqslant_{\mathrm{CT}} \mathtt{ct}_1 \quad \mathtt{ct} \leqslant_{\mathrm{CT}} \mathtt{ct}_2}{\mathtt{ct} \leqslant_{\mathrm{CT}} \mathtt{ct}_1 \wedge \mathtt{ct}_2}$$

$$[\text{type}]\ \frac{\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2}{\mathtt{t}_1 \leqslant_{\mathrm{CT}} \mathtt{t}_2}$$

Fig. 3. Coercions between conjunction types.

Since a term can only have one underlying type a well-formed conjunction type will not involve types with different underlying types. The well-formedness predicate is defined by:

$$\frac{}{\vdash^W \mathtt{t}}$$

$$\frac{\vdash^W \mathtt{ct}_1 \quad \vdash^W \mathtt{ct}_2}{\vdash^W \mathtt{ct}_1 \wedge \mathtt{ct}_2} \quad \text{if } \varepsilon(\mathtt{ct}_1) = \varepsilon(\mathtt{ct}_2)$$

This allows us to overload the function ε to also find the underlying type of a conjunction type: $\varepsilon(\mathtt{ct}_1 \wedge \mathtt{ct}_2) = \varepsilon(\mathtt{ct}_1)$. The predicate BOT is lifted to conjunction types:

$$\mathrm{BOT}(\mathtt{ct}_1 \wedge \mathtt{ct}_2) = \mathrm{BOT}(\mathtt{ct}_1) \wedge \mathrm{BOT}(\mathtt{ct}_2)$$
$$\mathrm{BOT}(\mathtt{t}) = \mathrm{BOT}(\mathtt{t})$$

The rules for coercing between conjunction types are given in Fig. 3. The relation $\leqslant_{\mathrm{CT}}$ is reflexive and transitive, and for strictness and totality types the relation is inherited from the relation, $\leqslant_{\mathrm{ST}}$, on strictness and totality types; this is expressed by the rule [type] in Fig. 3. For conjunctions we have three rules enforcing that "$\wedge$" is the greatest lower bound wrt $\leqslant_{\mathrm{ST}}$. However we do not have the rule

$$\frac{}{(\mathtt{t}_1 \to \mathtt{t}_2) \wedge (\mathtt{t}_1 \to \mathtt{t}_3) \leqslant_{\mathrm{CT}} \mathtt{t}_1 \to (\mathtt{t}_2 \wedge \mathtt{t}_3)}$$

because $(\mathtt{t}_1 \to (\mathtt{t}_2 \wedge \mathtt{t}_3))$ is not a well-formed top-level-conjunction type.

2.4. The conjunction type system

We have now prepared the ground for presenting the conjunction type inference system of Fig. 4. The list A of assumptions gives strictness and totality types to the free variables and again each variable may only occur once in the list. As mentioned

$$[\text{var}]\ \frac{}{A \vdash_{ST} x : t}\quad \text{if } (x : t) \in A$$

$$[\text{abs}]\ \frac{A, x : t_1 \vdash_{ST} e : t_2}{A \vdash_{ST} \lambda x.e : t_1 \to t_2}$$

$$[\text{abs2}]\ \frac{A, x : t_1 \vdash_{ST} e : t_2}{A \vdash_{ST} \lambda x.e : \varepsilon(t_1 \to t_2)^{\mathbf{n}}}$$

$$[\text{app}]\ \frac{A \vdash_{ST} e_1 : t_1 \to t_2 \quad A \vdash_{ST} e_2 : t_1}{A \vdash_{ST} e_1\ e_2 : t_2}$$

$$[\text{if1}]\ \frac{A \vdash_{ST} e_1 : \mathtt{Bool}^{\mathbf{b}} \quad A \vdash_{ST} e_2 : ct \quad A \vdash_{ST} e_3 : ct}{A \vdash_{ST} \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : \varepsilon(ct)^{\mathbf{b}}}$$

$$[\text{if2}]\ \frac{A \vdash_{ST} e_1 : \mathtt{Bool}^{\mathbf{n}} \quad A \vdash_{ST} e_2 : ct \quad A \vdash_{ST} e_3 : ct}{A \vdash_{ST} \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : ct}$$

$$[\text{if3}]\ \frac{A \vdash_{ST} e_1 : \mathtt{Bool}^{\top} \quad A \vdash_{ST} e_2 : ct \quad A \vdash_{ST} e_3 : ct}{A \vdash_{ST} \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : ct}\quad \text{if BOT}(ct)$$

$$[\text{fix}]\ \frac{A \vdash_{ST} e : (t_0 \to t_1) \wedge (t_1 \to t_2) \wedge \ldots \wedge (t_{n-1} \to t_n)}{A \vdash_{ST} \mathtt{fix}\ e : t_n}\quad \text{if } \begin{cases} \text{BOT}(t_0), \\ \exists p,q : p < q \\ \wedge t_q \leqslant_{ST} t_p \end{cases}$$

$$[\text{const}]\ \frac{}{A \vdash_{ST} c : ct_c}$$

$$[\text{coer}]\ \frac{A \vdash_{ST} e : ct_1}{A \vdash_{ST} e : ct_2}\quad \text{if } ct_1 \leqslant_{CT} ct_2$$

$$[\text{conj}]\ \frac{A \vdash_{ST} e : ct_1 \quad A \vdash_{ST} e : ct_2}{A \vdash_{ST} e : ct_1 \wedge ct_2}$$

Fig. 4. Conjunction type inference.

previously we assume that all the variables in the list are distinct. Only the lambda abstraction can extend the assumption list and since conjunction types only can appear at the top-level this means that assumption lists always will associate strictness and totality types, not conjunction types, with the variables. For each constant c, we assume that a conjunction type ct_c is specified; as an example $ct_{succ} = (\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$ where succ is the successor function.

The rules [var], [abs], [app], and [const] are just as their standard type inference counterparts in Fig. 1. There are three rules for conditional — depending on whether the test is of strictness and totality type $\mathtt{Bool}^{\mathbf{b}}$, $\mathtt{Bool}^{\mathbf{n}}$, or $\mathtt{Bool}^{\top}$.

The rule [coer] allows to change the strictness and totality type to one that is greater. It is quite useful as a preparation for applying rules [if2] and [if3]. The rule [conj] facilitates the construction of conjunction types (as is the case also for rule [const]).

From rule [fix] we may derive rules

$$[\text{fix1}]\ \frac{A \vdash_{ST} e : t \to t}{A \vdash_{ST} \mathtt{fix}\ e : t}\quad \text{if BOT}(t)$$

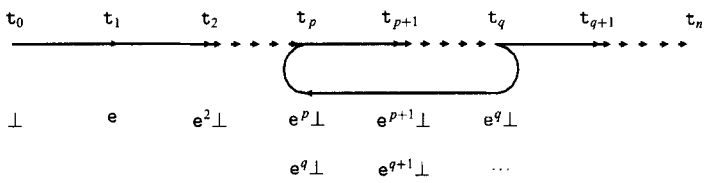

Fig. 5. Picturing the [fix]-rule.

and

$$[\text{fix2}]\ \frac{A \vdash_{ST} e : t_1 \to t_2}{A \vdash_{ST} \mathtt{fix}\ e : t_2} \quad \text{if } \mathrm{BOT}(t_1) \text{ and } t_2 \leqslant_{ST} t_1$$

that are simpler and perhaps more intuitive. For a comparison of the three fix rules see Appendix A. Note that in rule [fix] we have to ensure that the type t_0 can describe bottom in order to be able to calculate the fixpoint. After the first iteration, see Fig. 5, the term has the strictness and totality type t_1 and after the second the strictness and totality type t_2, etc. When the term reaches the strictness and totality type t_q we can apply the rule [coer] because we have $t_q \leqslant_{ST} t_p$ and so the term has the strictness and totality type t_p. In this way we can go on as long as necessary "to evaluate the fixpoint". Finally we iterate $n - q$ more times to get the type t_n for the fixpoint.

The following observations are easily verified by induction on the shape of the inference tree:

Fact 6. *If* $A \vdash_{ST} e : ct$ *then* $\vdash^W ct$ *and* $\varepsilon(A) \vdash e : \varepsilon(ct)$.

We also have a form of completeness:

Fact 7. *If* $A \vdash e : ut$ *then* $\mathtt{top}(A) \vdash_{ST} e : ut^\top$ *where* $\mathtt{top}(x : ut, A) = (x : ut^\top)$, $\mathtt{top}(A)$.

Example 8. In the inference system we can infer $\emptyset \vdash_{ST} \mathtt{fix}\ (\lambda x.x) : \mathtt{Int}^{\mathbf{b}}$ which is more precise than the $\mathtt{Int}^\top$ obtained by [22]. In the systems of [2, 6, 8] the best one can infer is the type $\mathtt{Int}^\top$ for the term $\mathtt{fix}\ (\lambda x.7)$ whereas we can infer $\emptyset \vdash_{ST} \mathtt{fix}\ (\lambda x.7) : \mathtt{Int}^{\mathbf{n}}$ and so again are more precise.

Example 9. The term `twice` is given by

$$\lambda f.\lambda x.f\ (f\ x)$$

has the strictness and totality type

$$t = (\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}}) \to \mathtt{Int}^\top \to \mathtt{Int}^{\mathbf{b}}$$

In order to show this we need to apply the rule [monotone]. For this let

$$A = \mathtt{f} : \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}}, \mathtt{x} : \mathtt{Int}^{\top}$$

and let P_1 be

$$\frac{\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \leqslant_{ST} \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}}{A \vdash_{ST} \mathtt{f} : \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}} \ [\text{var}] + [\text{coer}] + [\text{monotone}]$$

and let P_2 be

$$\frac{\dfrac{\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \leqslant_{ST} \mathtt{Int}^{\top} \to \mathtt{Int}^{\top}}{A \vdash_{ST} \mathtt{f} : \mathtt{Int}^{\top} \to \mathtt{Int}^{\top}} \ [\text{var}] + [\text{coer}] \qquad A \vdash_{ST} \mathtt{x} : \mathtt{Int}^{\top}}{A \vdash_{ST} \mathtt{f\ x} : \mathtt{Int}^{\top}}$$

Now we have

$$\frac{\dfrac{\dfrac{P_1 \qquad P_2}{A \vdash_{ST} \mathtt{f\ (f\ x)} : \mathtt{Int}^{\mathbf{b}}} \ [\text{app}]}{\mathtt{f} : \mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{b}} \vdash_{ST} \lambda \mathtt{x.f\ (f\ x)} : \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}} \ [\text{abs}]}{\emptyset \vdash_{ST} \lambda \mathtt{f}.\lambda \mathtt{x.f\ (f\ x)} : \mathtt{t}} \ [\text{abs}]$$

In this example we have used the rule [monotone] in an essential way.

Example 10. Consider the term[1] e and types $\mathtt{t}_1$ and $\mathtt{t}_2$:

$$\mathtt{e} = \lambda \mathtt{f}.\lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z.if\ (=\ 0\ z)\ then\ +\ x\ y\ else\ f\ y\ x\ (-\ z\ 1)}$$
$$\mathtt{t}_1 = \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$
$$\mathtt{t}_2 = \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$

We want to infer $\emptyset \vdash_{ST} \mathtt{fix\ e} : \mathtt{t}_1$ but it is not possible to infer $\emptyset \vdash_{ST} \mathtt{e} : \mathtt{t}_1 \to \mathtt{t}_1$. However we can infer $\emptyset \vdash_{ST} \mathtt{e} : \mathtt{t}_1 \to \mathtt{t}_2$ and $\emptyset \vdash_{ST} \mathtt{e} : \mathtt{t}_2 \to \mathtt{t}_1$ and we can apply the [conj]-rule to get $\emptyset \vdash_{ST} \mathtt{e} : (\mathtt{t}_1 \to \mathtt{t}_2) \wedge (\mathtt{t}_2 \to \mathtt{t}_1)$. Now we are able to apply the rule [fix] and thereby get $\emptyset \vdash_{ST} \mathtt{fix\ e} : \mathtt{t}_1$ as desired. This shows that even though we do not have "full" conjunction system of Jensen and Benton [7, 2] we *can* make good use of conjunction to type the "difficult" example of [9].

Example 11. Consider next the term[2] e given by

$$\mathtt{e} = \mathtt{twice\ g}$$
$$\mathtt{twice} = \lambda \mathtt{f}.\lambda \mathtt{x.f\ (f\ x)}$$
$$\mathtt{g} = \lambda \mathtt{y}.\lambda \mathtt{x.+\ x\ (y\ (fix\ \lambda x.x))}$$

[1] This example is due to Kuo and Mishra [9].

[2] Thanks to Nick Benton for pointing to this example.

It will have the strictness and totality type $(\mathtt{Int}^{\top} \to \mathtt{Int}^{\top}) \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$ but we are *not* able to obtain it using our analysis because one needs full conjunction in order to construct the proof-tree. The reason is that we need to infer that `twice` has the type

$$((\mathtt{t}_1 \to \mathtt{t}_2) \wedge (\mathtt{t}_2 \to \mathtt{t}_3)) \to (\mathtt{t}_1 \to \mathtt{t}_3)$$

for any $\mathtt{t}_1$, $\mathtt{t}_2$, and $\mathtt{t}_3$ but this is not a well-formed conjunction type in the current system.

3. Operational semantics

The first step towards showing the analysis sound is to define the semantics. The semantics will be lazy except that all built-in functions will be strict in each argument. Fig. 6 defines a natural-style operational semantics [15]. Terms are evaluated to WHNF, i.e. to constants or lambda-abstractions; we will let u, v, c, and f be such WHNFs. The meaning of a constant c is given by a set $\delta(\mathtt{c})$ of pairs of constants; the idea is that if $(\mathtt{u}, \mathtt{v}) \in \delta(\mathtt{c})$ then $\mathtt{c\ u} = \mathtt{v}$; e.g. $(2, +_2) \in \delta(+)$ and $(1, 3) \in \delta(+_2)$. As mentioned in the introduction to this paper the semantics is faithful to current lazy languages like Miranda [19] and this is unlike other approaches (e.g. [3]) where terms are evaluated to HNF rather than WHNF. As usual we shall regard α-equivalent terms as being equal.

Two closed terms are semantically equivalent, written $\mathtt{e}_1 \sim_{\mathtt{ut}} \mathtt{e}_2$, if they evaluate to the *same* WHNF and have the same underlying type:

Definition 12

$$(\mathtt{e}_1 \sim_{\mathtt{ut}} \mathtt{e}_2) \Leftrightarrow ((\vdash \mathtt{e}_1 \Downarrow \mathtt{v}) \Leftrightarrow (\vdash \mathtt{e}_2 \Downarrow \mathtt{v}))$$

provided both $\emptyset \vdash \mathtt{e}_1 : \mathtt{ut}$ and $\emptyset \vdash \mathtt{e}_2 : \mathtt{ut}$ can be inferred.

$$[\mathrm{app1}]\ \frac{\vdash \mathtt{e}_1 \Downarrow \lambda\mathtt{x.e} \quad \vdash \mathtt{e}[\mathtt{e}_2/\mathtt{x}] \Downarrow \mathtt{v}}{\vdash \mathtt{e}_1\ \mathtt{e}_2 \Downarrow \mathtt{v}}$$

$$[\mathrm{app2}]\ \frac{\vdash \mathtt{e}_1 \Downarrow \mathtt{c} \quad \vdash \mathtt{e}_2 \Downarrow \mathtt{v}}{\vdash \mathtt{e}_1\ \mathtt{e}_2 \Downarrow \mathtt{u}}\ \text{if } (\mathtt{v}, \mathtt{u}) \in \delta(\mathtt{c})$$

$$[\mathrm{fix}]\ \frac{\vdash \mathtt{e}\ (\mathtt{fix\ e}) \Downarrow \mathtt{v}}{\vdash \mathtt{fix\ e} \Downarrow \mathtt{v}}$$

$$[\mathrm{abs}]\ \frac{}{\vdash \lambda\mathtt{x.e} \Downarrow \lambda\mathtt{x.e}} \qquad [\mathrm{const}]\ \frac{}{\vdash \mathtt{c} \Downarrow \mathtt{c}}$$

$$[\mathrm{condT}]\ \frac{\vdash \mathtt{e}_1 \Downarrow \mathtt{true} \quad \vdash \mathtt{e}_2 \Downarrow \mathtt{v}_2}{\vdash \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3 \Downarrow \mathtt{v}_2}$$

$$[\mathrm{condF}]\ \frac{\vdash \mathtt{e}_1 \Downarrow \mathtt{false} \quad \vdash \mathtt{e}_3 \Downarrow \mathtt{v}_3}{\vdash \mathtt{if\ e}_1\ \mathtt{then\ e}_2\ \mathtt{else\ e}_3 \Downarrow \mathtt{v}_3}$$

Fig. 6. Lazy semantics for closed terms.

We shall assume throughout this paper that there are no empty types, i.e. for each underlying type there exists a *terminating* term of that type. Clearly, for each type there exists a non-terminating term of that type, for example $\mathtt{fix}\ (\lambda \mathtt{x}.\mathtt{x})$.

We shall write $\vdash \mathtt{e} \not\Downarrow$ to mean $\neg(\exists \mathtt{v} : \vdash \mathtt{e} \Downarrow \mathtt{v})$; this means that e does not terminate.

3.1. New terms

For the proof of soundness of the conjunction inference system we find it helpful to introduce the terms $\mathtt{fix}_n\ \mathtt{e}$ where n is a number greater than or equal to 0. The idea is that n indicates how many times the fixpoint is allowed to be unfolded. So we need to expand the underlying type inference system and the semantics of the simply-typed λ-calculus. The underlying type of $\mathtt{fix}_n\ \mathtt{e}$ is the same as for $\mathtt{fix}\ \mathtt{e}$:

$$[\mathrm{fix}_n]\ \frac{\mathtt{A} \vdash \mathtt{e} : \mathtt{ut} \to \mathtt{ut}}{\mathtt{A} \vdash \mathtt{fix}_n\ \mathtt{e} : \mathtt{ut}}$$

and the semantics for $\mathtt{fix}_n\ \mathtt{e}$ is:

$$[\mathrm{fix}_n]\ \frac{\vdash \mathtt{e}\ (\mathtt{fix}_n\ \mathtt{e}) \Downarrow \mathtt{v}}{\vdash \mathtt{fix}_{n+1}\ \mathtt{e} \Downarrow \mathtt{v}}$$

There are no rules for $\mathtt{fix}_0\ \mathtt{e}$ and hence $\mathtt{fix}_0\ \mathtt{e}$ is stuck. We will allow the function ε to be applied to a term to remove all the annotations on fix. We do not allow the programmer to use $\mathtt{fix}_n$; hence there is no need for analysis of terms including $\mathtt{fix}_j$; it is merely a piece of syntax needed to facilitate the proof of the soundness theorem.

For proving the monotonicity-rule sound we need to construct a terminating term given any term e in such a way that the new term computes the same WHNF as e and terminates if e loops. However, this new term must also terminate when applied to a number of arguments. Consider the term $\lambda \mathtt{x}.\mathtt{x}$ which evaluates to $\lambda \mathtt{x}.\mathtt{x}$. Now we want that the new term associated with $\lambda \mathtt{x}.\mathtt{x}$ applied to any argument terminates even if $\lambda \mathtt{x}.\mathtt{x}$ applied to the same argument does not terminate. To achieve this we introduce the new terms $\mathscr{T}_{\mathtt{e}}^{n}$ where e is a closed term without any $\mathtt{fix}_j$. The idea is that $\mathscr{T}_{\mathtt{e}}^{n}$ terminates when applied to $i \leqslant n$ arguments. The underlying type of $\mathscr{T}_{\mathtt{e}}^{n}$ is the same as for e:

$$[\mathscr{T}^n]\ \frac{\mathtt{A} \vdash \mathtt{e} : \mathtt{ut}}{\mathtt{A} \vdash \mathscr{T}_{\mathtt{e}}^{n} : \mathtt{ut}}$$

Let the arity of a standard type be 0 for base-types and for the function type, $\mathtt{ut}_1 \to \mathtt{ut}_2$, it is 1 plus the arity of $\mathtt{ut}_2$ and the *final result type* for a base-type B is B and for the function type, $\mathtt{ut}_1 \to \mathtt{ut}_2$, it is the final final result type of $\mathtt{ut}_2$. Now the semantics for $\mathscr{T}_{\mathtt{e}}^{n}$ is:

$$[\mathrm{eval1}]\ \frac{\vdash \mathtt{e} \Downarrow \mathtt{v}}{\vdash \mathscr{T}_{\mathtt{e}}^{0} \Downarrow \mathtt{v}}$$

$$[\mathrm{eval2}]\ \frac{\vdash \mathtt{e} \Downarrow \mathtt{v}}{\vdash \mathscr{T}_{\mathtt{e}}^{n+1} \Downarrow \mathscr{T}_{\mathtt{v}}^{n+1}}$$

$$[\text{eval3}]\ \frac{\vdash \mathtt{e} \Downarrow}{\vdash \mathscr{T}_{\mathtt{e}}^{n} \Downarrow \lambda \mathtt{x}_1 \ldots . \ \lambda \mathtt{x}_a . \ \mathtt{c}_{\mathtt{B}}} \quad \text{if} \begin{cases} \emptyset \vdash \mathtt{e} : \mathtt{ut} \\ a \text{ is the arity of } \mathtt{ut} \text{ and} \\ \mathtt{B} \text{ is the final result type of } \mathtt{ut} \end{cases}$$

$$[\mathscr{T}^{n}\text{app}]\ \frac{\vdash \mathtt{e}_1 \Downarrow \mathscr{T}_{\mathtt{V}}^{n+1} \quad \vdash \mathscr{T}^{n}_{\mathtt{V}\varepsilon(\mathtt{e}_2)} \Downarrow \mathtt{v}'}{\vdash \mathtt{e}_1\ \mathtt{e}_2 \Downarrow \mathtt{v}'}$$

where $\mathtt{c}_{\mathtt{B}}$ is a constant of type B. Again the programmer is not allowed to use terms including $\mathscr{T}^n$; they are only introduced to be used in the soundness proof of the analysis.

The reason for not allowing terms to include $\mathtt{fix}_j$ inside the annotation on $\mathscr{T}^n$ is that otherwise monotonicity of evaluation will not be preserved. Consider Fact 17, below, and the term $\mathtt{fix}_6\ \mathtt{fac}\ \ 7$. We have

$$\vdash \mathtt{fix}_6\ \mathtt{fac}\ \ 7 \not\Downarrow$$

and by the [eval3] rule we get

$$\vdash \mathscr{T}^{0}_{\mathtt{fix}_6\ \mathtt{fac}\ 7} \Downarrow \mathtt{c}_{\mathtt{Int}}$$

However we have

$$\vdash \varepsilon(\mathscr{T}^{0}_{\mathtt{fix}_6\ \mathtt{fac}\ 7}) \Downarrow 5040$$

3.2. *Properties of the semantics*

Whenever $\mathtt{e}_1$ does not evaluate, then $\mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3$ cannot evaluate either:

Fact 13

$$\vdash \mathtt{e}_1 \not\Downarrow \Rightarrow \vdash \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3 \not\Downarrow$$

Whenever the function does not evaluate the application cannot evaluate either:

Fact 14

$$\vdash \mathtt{e} \not\Downarrow \Rightarrow \vdash \mathtt{e}\ \mathtt{e}' \not\Downarrow$$

Provided $\mathtt{e}_1$ and $\mathtt{e}_2$ are semantically equivalent, then $(\mathtt{e}_1\ \mathtt{e}')$ and $(\mathtt{e}_2\ \mathtt{e}')$ are semantically equivalent:

Fact 15

$$(\mathtt{e}_1 \sim_{\mathtt{ut}_1 \to \mathtt{ut}_2} \mathtt{e}_2) \Rightarrow (\mathtt{e}_1 \mathtt{e}' \sim_{\mathtt{ut}_2} \mathtt{e}_2\ \mathtt{e}')$$

The underlying types that can be inferred for a term e without any $\texttt{fix}_n$'s can also be inferred for the term e′ with $\texttt{fix}_n$ replacing some occurrences of fix and vice versa:

Fact 16

$$(A \vdash \texttt{e} : \texttt{ut}) \Leftrightarrow (A \vdash \varepsilon(\texttt{e}) : \texttt{ut})$$

3.2.1. Fixpoints

A fixpoint that can evaluate with n unfoldings can also evaluate if it is allowed to unfold an unlimited number of times:

Fact 17

$$(\vdash \texttt{e} \Downarrow \texttt{v}) \Rightarrow (\vdash \varepsilon(\texttt{e}) \Downarrow \varepsilon(\texttt{v}))$$

We now show that if (fix e) evaluates then there exists a number n such that ($\texttt{fix}_n$ e) evaluates. In the proof of this result we need a way to modify some of the occurrences of fix in a term. For this we introduce the notion of tree-substitutions, π. They will tell us which occurrences of fix to replace with an occurrence of $\texttt{fix}_j$.

Definition 18 (*Tree-substitution*). A tree-substitution π is a set of pairs of tree-addresses and a number. A tree-address is a list of 0, 1, 2.

For $n \in \{0, 1, 2\}$ let π^n be the part of the tree-substitution π where all the tree-addresses starts with an n but without this leading n, i.e.

$$\pi^n = \{(\text{addr}, m) \mid (n : \text{addr}, m) \in \pi\}$$

where "n : addr" denotes the list whose first element is n and whose tail is addr. Let $\pi + n$ be the tree-substitution

$$\pi + n = \{(\text{addr}, m + n) \mid (\text{addr}, m) \in \pi\}.$$

Let $n\pi$ be the tree-substitution

$$n\pi = \{(n : \text{addr}, m) \mid (\text{addr}, m) \in \pi\}$$

and let $p\pi$ be the tree-substitution

$$\{(p \mathbin{++} \text{addr}, m) \mid (\text{addr}, m) \in \pi \wedge p \text{ is a tree-address}\}$$

where "++" denotes list concatenation.

The tree-substitution π applied to a term e is written $[\mathtt{e}]^{\pi}$ and is defined inductively as follows:

$$
\begin{aligned}
[\mathtt{x}]^{\pi} &= \mathtt{x} \\
[\mathtt{c}]^{\pi} &= \mathtt{c} \\
[\mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3]^{\pi} &= \mathtt{if}\ [\mathtt{e}_1]^{\pi^0}\ \mathtt{then}\ [\mathtt{e}_2]^{\pi^1}\ \mathtt{else}\ [\mathtt{e}_3]^{\pi^2} \\
[\mathtt{e}_1\ \mathtt{e}_2]^{\pi} &= [\mathtt{e}_1]^{\pi^0}\ [\mathtt{e}_2]^{\pi^1} \\
[\lambda \mathtt{x}.\mathtt{e}]^{\pi} &= \lambda \mathtt{x}.[\mathtt{e}]^{\pi^0} \\
[\mathtt{fix}\ \mathtt{e}]^{\pi} &= \begin{cases} \mathtt{fix}_n\ [\mathtt{e}]^{\pi^0}, & \text{if } ([\,], n) \in \pi \\ \mathtt{fix}\ [\mathtt{e}]^{\pi^0}, & \text{otherwise} \end{cases} \\
[\mathscr{T}_{\mathtt{e}}^{n}]^{\pi} &= \mathscr{T}_{\mathtt{e}}^{n}
\end{aligned}
$$

where [ ] denotes the empty list.

Proposition 19. *For* e *without any* $\mathtt{fix}_j$ *we have*

$$
\vdash \mathtt{e} \Downarrow \mathtt{v} \Rightarrow \begin{cases} \forall \pi \exists m \exists \pi' \forall n \geqslant 0: \\ \quad (\vdash [\mathtt{e}]^{\pi+m+n} \Downarrow [\mathtt{v}]^{\pi'+n}) \wedge \\ \quad ((\vdash [\mathtt{e}]^{\pi} \Downarrow \mathtt{v}') \Rightarrow m = 0) \end{cases}
$$

The idea is that for any labelling of the fixpoints in a term, m is the minimal number to be added so that the term can evaluate. The number n indicates that whenever a labelling of the fixpoints will let the term evaluate, then increasing the labels it will still let the term evaluate. This is stated in Corollary 21 below.

Proof. We assume $\vdash \mathtt{e} \Downarrow \mathtt{v}$ and that e is without any $\mathtt{fix}_j$; then we prove by induction in the proof-tree for $\vdash \mathtt{e} \Downarrow \mathtt{v}$ that

$$
\begin{aligned}
&\forall \pi \exists m \exists \pi' \forall n \geqslant 0: \\
&\quad (\vdash [\mathtt{e}]^{\pi+m+n} \Downarrow [\mathtt{v}]^{\pi'+n}) \wedge ((\vdash [\mathtt{e}]^{\pi} \Downarrow \mathtt{v}') \Rightarrow m = 0)
\end{aligned}
$$

holds. The proof is rather technical and involves keeping track of the $\mathtt{fix}_j$ terms during evaluation. We illustrate one typical case below and refer to [17] for the full details.

The case [app1]: We assume $\vdash \mathtt{e}_1\ \mathtt{e}_2 \Downarrow \mathtt{v}$ and that $\mathtt{e}_1\ \mathtt{e}_2$ is without any $\mathtt{fix}_j$. From the [app1]-rule we get $\vdash \mathtt{e}_1 \Downarrow \lambda \mathtt{x}.\mathtt{e}$ and $\vdash \mathtt{e}[\mathtt{e}_2/\mathtt{x}] \Downarrow \mathtt{v}$. By applying the induction hypothesis we get

$$
\begin{aligned}
&\forall \pi_1 \exists m_1 \exists \pi_1' \forall n_1 \geqslant 0: \\
&\quad (\vdash [\mathtt{e}_1]^{\pi_1+m_1+n_1} \Downarrow [\lambda \mathtt{x}.\mathtt{e}]^{\pi_1'+n_1}) \wedge && (2) \\
&\quad (\vdash [\mathtt{e}_1]^{\pi_1} \Downarrow \mathtt{v}_1') \Rightarrow m_1 = 0) && (3)
\end{aligned}
$$

$$\forall\pi_2\exists m_2\exists\pi_2'\forall n_2\geqslant 0:$$

$$(\vdash[\mathtt{e}[\mathtt{e}_2/\mathtt{x}]]^{\pi_2+m_2+n_2}\Downarrow[\mathtt{v}]^{\pi_2'+n_2})\wedge \tag{4}$$

$$((\vdash[\mathtt{e}[\mathtt{e}_2/\mathtt{x}]]^{\pi_2}\Downarrow\mathtt{v}_2')\Rightarrow m_2=0) \tag{5}$$

Now we let $\pi_1 = \pi^0$ and from (2) we get m_1 and π_1' such that for all $m_2\geqslant 0$ and $n\geqslant 0$ we have

$$\vdash[\mathtt{e}_1]^{\pi^0+m_1+m_2+n}\Downarrow\lambda\mathtt{x}.[\mathtt{e}]^{\pi_1'^0+n+m_2}. \tag{6}$$

Now we find all the addresses of x in e and concatenate them to π^1; that is

$$\pi_2=\pi'^0\cup\{p\pi^1+m_1\mid p \text{ is a tree-address of } \mathtt{x} \text{ in } \mathtt{e}\}.$$

From (4) we get m_2 and π_2' such that for all $n\geqslant 0$ we have

$$\vdash[\mathtt{e}]^{\pi'^0+m_2+n}[[\mathtt{e}_2]^{\pi^1+m+n}/\mathtt{x}]\Downarrow[\mathtt{v}]^{\pi_2'+n} \tag{7}$$

We now set $m = m_1 + m_2$ and $\pi' = \pi_2'$ and by applying the rule [app1] to (6) and (7) we get $\vdash[\mathtt{e}_1.\mathtt{e}_2]^{\pi+m+n}\Downarrow[\mathtt{v}]^{\pi_2'+n}$ for all $n\geqslant 0$ as required for the first part.

For the second part we assume $\vdash[\mathtt{e}_1\ \mathtt{e}_2]^{\pi}\Downarrow\mathtt{v}'$, i.e. $\vdash[\mathtt{e}_1]^{\pi^0}\ [\mathtt{e}_2]^{\pi^1}\Downarrow\mathtt{v}'$. From the [app1]-rule we get $\vdash[\mathtt{e}_1]^{\pi^0}\Downarrow\lambda\mathtt{x}'.\mathtt{e}'$ and $\vdash\mathtt{e}'[[\mathtt{e}_2]^{\pi^1}/\mathtt{x}']\Downarrow\mathtt{v}'$. From (3) we get $m_1 = 0$ hence by (2) we have $\lambda\mathtt{x}'.\mathtt{e}' = [\lambda\mathtt{x}.\mathtt{e}]^{\pi_1'}$. Furthermore we have $\mathtt{e}'[[\mathtt{e}_2]^{\pi^1}/\mathtt{x}'] = [\mathtt{e}[\mathtt{e}_2/\mathtt{x}]]^{\pi_2}$ and $\mathtt{v}' = [\mathtt{v}]^{\pi_2'}$. From (4) we have $m_2 = 0$ and we have $m = m_1 + m_2 = 0 + 0 = 0$ as required. □

Corollary 20. $(\vdash\mathtt{fix}\ \mathtt{e}\Downarrow\mathtt{v})\Rightarrow(\exists m\exists\mathtt{v}':\vdash\mathtt{fix}_m\ \mathtt{e}\Downarrow\mathtt{v}')$ *provided* e *is without any* $\mathtt{fix}_j$.

Proof. Use Proposition 19 with $\pi = \{([\ ], 0)\}$ and $n = 0$. □

A fixpoint that can evaluate with k unfoldings can also evaluate if it is allowed to unfold $k + 1$ times:

Corollary 21. $(\vdash\mathtt{fix}_k\ \mathtt{e}\Downarrow\mathtt{v})\Rightarrow(\exists\mathtt{v}':\vdash\mathtt{fix}_{k+1}\ \mathtt{e}\Downarrow\mathtt{v}')$ *provided* e *is without any* $\mathtt{fix}_j$.

Proof. Use Proposition 19 with $\pi = \{([\ ], k)\}$ and $n = 1$ and observe that $m = 0$. □

3.2.2. Terminating terms

Suppose that a term e applied to some terms does indeed evaluate; we now consider to which term $\mathscr{T}^n_{\varepsilon(\mathtt{e})}$ evaluates when applied to the same terms.

Lemma 22. *Given* $1\leqslant i\leqslant n\leqslant a$ *where a is the arity of* e:

$$(\vdash\ \mathtt{e}\ \mathtt{e}_1\ \ldots\ \mathtt{e}_i\Downarrow\mathtt{v})\Rightarrow\left(\vdash\ \mathscr{T}^n_{\varepsilon(\mathtt{e})}\ \mathtt{e}_1\ \ldots\ \mathtt{e}_i\Downarrow\begin{cases}\varepsilon(\mathtt{v}), & \text{if } n=i\\ \mathscr{T}^{n-i}_{\varepsilon(\mathtt{v})}, & \text{otherwise}\end{cases}\right)$$

Proof. The lemma is shown by induction on i. □

Next suppose that a term e does not evaluate when applied to certain terms; we now consider what happens for $\mathscr{T}^n_{\varepsilon(\mathsf{e})}$ when applied to the same terms.

Lemma 23. *Given* $1 \leqslant i \leqslant n \leqslant a$ *where a is the arity of* e:

$$(\vdash \mathsf{e}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_i \not\Downarrow) \Rightarrow (\exists\ \mathsf{v}' : (\vdash \mathscr{T}^n_{\varepsilon(\mathsf{e})}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_i \Downarrow \mathsf{v}'))$$

Proof. We observe that either $\vdash \varepsilon(\mathsf{e}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_i) \Downarrow \mathsf{v}'$ or $\vdash \varepsilon(\mathsf{e}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_i) \not\Downarrow$ must be the case. In the first case we use Lemma 22. In the second case we use the rules [eval2] and [eval3]. □

Finally, from the proof-tree for the term $(\mathsf{e}\ \mathsf{e}'\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k)$ we can construct a proof-tree for the term $(\mathsf{e}\ \mathscr{T}^n_{\varepsilon(\mathsf{e}')}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k)$:

Lemma 24

$$(\vdash \mathsf{e}\ \mathsf{e}'\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k \Downarrow \mathsf{v}) \Rightarrow \exists\ \mathsf{v}'\ :\ \vdash \mathsf{e}\ \mathscr{T}^n_{\varepsilon(\mathsf{e}')}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k \Downarrow \mathsf{v}'$$

Proof. In this proof we regard a proof-tree as having its root at the bottom. For the proof we assume that $\vdash \mathsf{e}\ \mathsf{e}'\ \mathsf{e}_1 \ldots \mathsf{e}_k \Downarrow \mathsf{v}$ and we prove that $\vdash \mathsf{e}\ \mathscr{T}^n_{\varepsilon(\mathsf{e}')}\ \mathsf{e}_1 \ldots \mathsf{e}_k \Downarrow \mathsf{v}'$. We do this by first constructing a template for the given proof-tree and then later use this template to construct the desired proof-tree. For an example see Example 25 below.

To construct the template we first remove the parts of the proof-tree that are above certain nodes by traversing the given proof-tree in a left-most-top-first manner. Let u_0 be e'. Now remove the parts of the proof-tree that are above the nodes of the form:

- $\vdash \mathsf{u}_i \Downarrow \mathsf{u}_{i+1}$ with no nodes below that is u_i applied to a number of terms. For later use we let k_i be 0 in this case.
- $\vdash \mathsf{u}_i\ \mathsf{e}'_1\ \ldots\ \mathsf{e}'_{k_i} \Downarrow \mathsf{u}_{i+1}$ with no nodes below that is u_i applied to a greater number of terms.

We continue in this way until there are no more parts of the proof-tree that can be removed.

The template can be constructed by copying all nodes from the proof-tree resulting from the above process. However, in nodes involving any u_i we replace u_i with a pointer to the pair

$$\left(\mathsf{u}_i, \begin{cases} \varepsilon(\mathsf{u}_i), & \text{if } n \leqslant k_0 + \cdots + k_i \\ \mathscr{T}^{n-k_0-\cdots-k_i}_{\varepsilon(\mathsf{u}_i)}, & \text{otherwise} \end{cases}\right)$$

Now note that the proof-tree for $\vdash \mathsf{e}\ \mathsf{e}'\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k \Downarrow \mathsf{v}$ may be constructed from the template by extracting the first component of the pairs and then constructing the top parts of the tree. In a similar way the proof-tree for $\vdash \mathsf{e}\ \mathscr{T}^n_{\varepsilon(\mathsf{e}')}\ \mathsf{e}_1\ \ldots\ \mathsf{e}_k \Downarrow \mathsf{v}$ is constructed

by extracting the second component of the pairs and using Lemma 22 to construct the top-parts of the tree. □

Example 25. Consider the term e e′, where

$e = \lambda x.+\ (+\ 1\ 2)\ (x\ 2)$

$e' = \lambda y.\times\ y\ 4$

The full evaluation-tree is:

$$\dfrac{\vdash e \Downarrow e \qquad \dfrac{\dfrac{\overline{\vdash + \Downarrow +} \quad P_1 \quad +_3 \in \delta(+,3) \qquad \overline{\vdash e' \Downarrow e'} \quad P_2}{\vdash +\ (+\ 1\ 2) \Downarrow +_3 \qquad \vdash e'\ 2 \Downarrow 8} \quad 11 \in \delta((+_3, 8))}{\vdash (+\ (+\ 1\ 2)\ (x\ 2))[e'/x] \Downarrow 11}}{\vdash e\ e' \Downarrow 11}$$

where P_1 is

$$\dfrac{\dfrac{\overline{\vdash + \Downarrow +} \qquad \overline{\vdash 1 \Downarrow 1} \quad +_1 \in \delta(+,1)}{\vdash +\ 1 \Downarrow +_1} \qquad \overline{\vdash 2 \Downarrow 2} \quad 3 \in \delta(+_1, 2)}{\vdash +\ 1\ 2 \Downarrow 3}$$

and P_2 is

$$\dfrac{\dfrac{\overline{\vdash \times \Downarrow \times} \qquad \overline{\vdash 2 \Downarrow 2} \quad \times_2 \in \delta(\times,2)}{\vdash \times\ 2 \Downarrow \times_2} \qquad \overline{\vdash 4 \Downarrow 4} \quad 8 \in \delta(\times_2, 4)}{\vdash (\times\ y\ 4)[2/y] \Downarrow 8}$$

First we remove the part of the tree that is above $\vdash e'\ 2 \Downarrow 8$ and it looks like

$$\dfrac{\vdash e \Downarrow e \qquad \dfrac{\dfrac{\overline{\vdash + \Downarrow +} \quad P_1 \quad +_3 \in \delta(+,3)}{\vdash +\ (+\ 1\ 2) \Downarrow +_3} \qquad \overline{\vdash e'\ 2 \Downarrow 8} \quad 11 \in \delta((+_3, 8))}{\vdash (+\ (+\ 1\ 2)\ (x\ 2))[e'/x] \Downarrow 11}}{\vdash e\ e' \Downarrow 11}$$

and we have

$u_0 = e'$

$u_1 = 8$

$u_2 = 11$

$k_0 = 1$

$k_1 = 0$

$k_2 = 0$

Now the template is:

$$\dfrac{\vdash e \Downarrow e \qquad \dfrac{\dfrac{\overline{\vdash + \Downarrow +} \quad P_1 \quad +_3 \in \delta(+,3)}{\vdash +\ (+\ 1\ 2) \Downarrow +_3} \qquad \overline{\vdash p_0\ 2 \Downarrow p_1} \quad p_2 \in \delta((+_3, p_1))}{\vdash (+\ (+\ 1\ 2)\ (x\ 2))[p_0/x] \Downarrow p_2}}{\vdash e\ p_0 \Downarrow p_2}$$

where

$$p_0 = \left(\mathsf{e}', \begin{cases} \mathsf{e}', & \text{if } n \leqslant 1 \\ \mathscr{T}_{\mathsf{e}'}^{n-1}, & \text{otherwise} \end{cases}\right)$$

$$p_1 = \left(8, \begin{cases} 8, & \text{if } n \leqslant 1 \\ \mathscr{T}_{8}^{n-1} s, & \text{otherwise} \end{cases}\right)$$

$$p_2 = \left(11, \begin{cases} 11, & \text{if } n \leqslant 1 \\ \mathscr{T}_{11}^{n-1}, & \text{otherwise} \end{cases}\right)$$

Whenever we want a proof-tree for $\mathsf{e}\ \mathsf{e}'$ we use the first component of the pairs and when we want a proof-tree for $\mathsf{e}\ \mathscr{T}_{\mathsf{e}'}^{n}$ we use the second component of the pairs.

4. Soundness

Our remaining task is to prove that the inference system of Fig. 4 is sound with respect to the natural-style operational semantics of Fig. 6. First we define a predicate $\models \mathsf{e} : \mathtt{ct}$ stating that the term e is valid of conjunction type ct. Then we show some useful lemmas and finally we prove the soundness result: if $\mathrm{A} \vdash_{\mathrm{ST}} \mathsf{e} : \mathtt{ct}$ then $\models \mathsf{e}[\overline{\mathsf{v}}/\overline{\mathsf{x}}] : \mathtt{ct}$ for all closed substitutions $[\overline{\mathsf{v}}/\overline{\mathsf{x}}]$ that are valid of the types in A.

The validity predicate is defined in Fig. 7. The term e is valid of conjunction type $\mathtt{ct}_1 \wedge \mathtt{ct}_2$ if e is valid of type $\mathtt{ct}_1$ as well as $\mathtt{ct}_2$. That the term e has a WHNF and the underlying type ut amounts to $\models \mathsf{e} : \mathtt{ut}^{\mathtt{n}}$ being true; that e has no WHNF but has the underlying type ut amounts to $\models \mathsf{e} : \mathtt{ut}^{\mathtt{b}}$ being true (i.e. there exists no WHNF, v, such that $\vdash \mathsf{e} \Downarrow \mathsf{v}$). A term with conjunction type $\mathtt{ut}^{\top}$ just has to be of the underlying type ut, as we do not know anything about the evaluation of the term. A term e is valid of function type $\mathtt{t}_1 \rightarrow \mathtt{t}_2$ if for any other term e' that is valid of strictness and totality type $\mathtt{t}_1$, also e applied to e' will be valid of strictness and totality type $\mathtt{t}_2$. (Here we see the importance of not having empty types as then the rule [notbot] would not be sound.)

$$\begin{array}{lll}
\text{(I)} & (\models \mathsf{e} : \mathtt{ct}_1 \wedge \mathtt{ct}_2) & \Leftrightarrow\ (\models \mathsf{e} : \mathtt{ct}_1) \wedge (\models \mathsf{e} : \mathtt{ct}_2) \\
\text{(II)} & (\models \mathsf{e} : \mathtt{ut}^{\mathtt{b}}) & \Leftrightarrow\ (\forall \mathsf{v}: \vdash \mathsf{e} \not\Downarrow) \wedge (\emptyset \vdash \mathsf{e} : \mathtt{ut}) \\
\text{(III)} & (\models \mathsf{e} : \mathtt{ut}^{\mathtt{n}}) & \Leftrightarrow\ (\exists \mathsf{v}: \vdash \mathsf{e} \Downarrow \mathsf{v}) \wedge (\emptyset \vdash \mathsf{e} : \mathtt{ut}) \\
\text{(IV)} & (\models \mathsf{e} : \mathtt{ut}^{\top}) & \Leftrightarrow\ (\emptyset \vdash \mathsf{e} : \mathtt{ut}) \\
\text{(V)} & (\models \mathsf{e} : \mathtt{t}_1 \rightarrow \mathtt{t}_2) & \Leftrightarrow\ (\forall \mathsf{e}': (\models \mathsf{e}' : \mathtt{t}_1) \Rightarrow (\models \mathsf{e}\ \mathsf{e}' : \mathtt{t}_2)) \\
 & & \quad \wedge (\emptyset \vdash \mathsf{e} : \varepsilon(\mathtt{t}_1) \rightarrow \varepsilon(\mathtt{t}_2))
\end{array}$$

Fig. 7. The definition of validity.

To prepare for the soundness of the conjunction type inference system we first need to bind all the free variables in the term. Let $\overline{\mathtt{x}}$ be the list of variables in Λ, let $\overline{\mathtt{t}}$ be the list of the strictness and totality types corresponding to the variables $\overline{\mathtt{x}}$, and let $\overline{\mathtt{v}}$ be a list of closed terms that are valid of the types $\overline{\mathtt{t}}$, i.e. $\models \overline{\mathtt{v}} : \overline{\mathtt{t}}$. Here we define $\models \overline{\mathtt{v}} : \overline{\mathtt{t}}$ inductively by

$$\models (\mathtt{v}, \overline{\mathtt{v}}) : (\mathtt{t}, \overline{\mathtt{t}}) = (\models \mathtt{v} : \mathtt{t}) \wedge (\models \overline{\mathtt{v}} : \overline{\mathtt{t}})$$

$$\models [\,] : [\,] = \mathtt{tt}$$

and the substitution $[\overline{\mathtt{v}}/\overline{\mathtt{x}}]$ is defined inductively by

$$\mathtt{e}[(\mathtt{v}, \overline{\mathtt{v}})/(\mathtt{x}, \overline{\mathtt{x}})] = (\mathtt{e}[\mathtt{v}/\mathtt{x}])[\overline{\mathtt{v}}/\overline{\mathtt{x}}]$$

$$\mathtt{e}[[\,]/[\,]] = \mathtt{e}$$

Theorem 26 (Soundness). *For expressions* e *without any* $\mathtt{fix}_n$ *and* $\mathscr{T}^n$ *we have*

$$(\overline{\mathtt{x}} : \overline{\mathtt{t}} \vdash_{\mathrm{ST}} \mathtt{e} : \mathtt{ct}) \Rightarrow (\forall\ \overline{\mathtt{v}} : (\models \overline{\mathtt{v}} : \overline{\mathtt{t}}) \Rightarrow (\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{ct})).$$

Before we prove the soundness theorem we need some facts and lemmas. They are divided into three groups: first we show a property of the underlying type system, then we show some properties of the analysis and finally we show some properties of the validity predicate.

4.1. Properties of the type systems

For a free variable x in a term e we can substitute a term e′ with the type indicated by the type environment A for x. The term e′ does not have to be closed but may only use the same free variables as e except for x.

Lemma 27

$$((\mathrm{A} \vdash \mathtt{e} : \mathtt{ut}_2) \wedge (\mathrm{A}_{\mathtt{x}} \vdash \mathtt{e}_2 : \mathtt{ut}_1) \wedge (\mathtt{x} : \mathtt{ut}_1 \in \mathrm{A})) \Rightarrow (\mathrm{A}_{\mathtt{x}} \vdash \mathtt{e}[\mathtt{e}_2/\mathtt{x}] : \mathtt{ut}_2)$$

Proof. We prove the lemma by straightforward induction in the proof-tree for the inference $\mathrm{A} \vdash \mathtt{e} : \mathtt{ut}_2$. For the full details see [17]. □

Two conjunction types can only be compared if they have the same underlying type:

Fact 28

$$(\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2) \Rightarrow (\varepsilon(\mathtt{t}_1) = \varepsilon(\mathtt{t}_2))$$

$$(\mathtt{ct}_1 \leqslant_{\mathrm{CT}} \mathtt{ct}_2) \Rightarrow (\varepsilon(\mathtt{ct}_1) = \varepsilon(\mathtt{ct}_2))$$

Now we list some properties of the function $\downarrow$:

Fact 29

(a) $\mathtt{t} \leqslant_{\mathrm{ST}} \downarrow\mathtt{t}$,
(b) $\downarrow(\downarrow\mathtt{t}) = \downarrow\mathtt{t}$,
(c) $(\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2) \Rightarrow (\downarrow\mathtt{t}_1 \leqslant_{\mathrm{ST}} \downarrow\mathtt{t}_2)$,
(d) $\mathrm{BOT}(\downarrow\mathtt{t}) = \mathtt{tt}$,
(e) $((\mathtt{t} \leqslant_{\mathrm{ST}} \mathtt{t}') \wedge (\mathrm{BOT}(\mathtt{t}')) \Rightarrow (\downarrow\mathtt{t} \leqslant_{\mathrm{ST}} \mathtt{t}')$.

Note that (e) expresses that $\downarrow\mathtt{t}$ is the smallest type such that both $\mathtt{t} \leqslant_{\mathrm{ST}} \downarrow\mathtt{t}$ and $\mathrm{BOT}(\downarrow\mathtt{t})$ holds.

Proof. Parts (a), (b) and (d) are proved by structural induction on $\mathtt{t}$ and part (e) by structural induction on $\mathtt{t}'$; part (c) is proved by induction on the structure of the inference $\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2$. □

Now we show that provided $\mathrm{BOT}(\mathtt{ct})$ is true, then the conjunction type $\mathtt{ct}$ is greater than $\varepsilon(\mathtt{ct})^{\mathbf{b}}$.

Lemma 30

$$(\mathrm{BOT}(\mathtt{ct}) = \mathtt{tt}) \Leftrightarrow (\varepsilon(\mathtt{ct})^{\mathbf{b}} \leqslant_{\mathrm{CT}} \mathtt{ct})$$

Proof. First we assume $\mathrm{BOT}(\mathtt{ct}) = \mathtt{tt}$ and then show by induction in the type $\mathtt{ct}$ that $\varepsilon(\mathtt{ct})^{\mathbf{b}} \leqslant_{\mathrm{CT}} \mathtt{ct}$ can be inferred. Second we assume $\varepsilon(\mathtt{ct})^{\mathbf{b}} \leqslant_{\mathrm{CT}} \mathtt{ct}$ and then show by induction the type $\mathtt{ct}$ that $\mathrm{BOT}(\mathtt{ct})$ is true. For the full details see [17]. □

4.2. Properties of the validity predicate

The term $\mathscr{T}_{\mathtt{e}}^{0}$ always terminates:

Lemma 31

$$(\models \mathtt{e} : \mathtt{ut}^{\top}) \Rightarrow (\models \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} : \mathtt{ut}^{\mathbf{n}})$$

Proof. We assume $\models \mathtt{e} : \mathtt{ut}^{\top}$. There are now two possibilities: either $\vdash \mathtt{e} \Downarrow \mathtt{v}$ or $\vdash \mathtt{e} \not\Downarrow$. First assume $\vdash \mathtt{e} \Downarrow \mathtt{v}$. From Fact 17 we have $\vdash \varepsilon(\mathtt{e}) \Downarrow \varepsilon(\mathtt{v})$ and from the rule [eval1] we get $\vdash \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} \Downarrow \varepsilon(\mathtt{v})$ hence we have $(\models \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} : \mathtt{ut}^{\mathbf{n}})$.

Secondly assume $\vdash \mathtt{e} \not\Downarrow$. Now it must either be the case that $\vdash \varepsilon(\mathtt{e}) \Downarrow \mathtt{v}$ or $\vdash \varepsilon(\mathtt{e}) \not\Downarrow$. In the first case we do as above and we have $(\models \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} : \mathtt{ut}^{\mathbf{n}})$. In the second case we apply the rule [eval3] to get $\vdash \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} \Downarrow \lambda\mathtt{x}_1 \ldots . \ \lambda\mathtt{x}_a . \mathtt{c}_{\mathtt{B}}$ where a is the arity of $\mathtt{ut}$ and $\mathtt{B}$ is the final result type of $\mathtt{ut}$. We now have $(\models \mathscr{T}^{0}_{\varepsilon(\mathtt{e})} : \mathtt{ut}^{\mathbf{n}})$ as required. □

The term $\mathscr{T}_{\mathtt{e}}^{n}$ applied to n terms will always terminate:

Lemma 32

$$(\models \mathtt{e} : \mathtt{t}_1 \to \ldots \mathtt{t}_n \to \mathtt{ut}^{\top}) \Rightarrow (\models \mathscr{T}^n_{\mathcal{E}(\mathtt{e})} : \mathtt{t}_1 \to \ldots \mathtt{t}_n \to \mathtt{ut}^{\mathbf{n}})$$

Proof. We assume $(\models \mathtt{e} : \mathtt{t}_1 \to \ldots \mathtt{t}_n \to \mathtt{ut}^{\top})$. We want to show

$$\models \mathscr{T}^n_{\mathcal{E}(\mathtt{e})} : \mathtt{t}_1 \to \ldots \mathtt{t}_n \to \mathtt{ut}^{\mathbf{n}}$$

which is equivalent to showing

$$\forall \mathtt{e}_1 \ldots \mathtt{e}_n : (\models \mathtt{e}_1 : \mathtt{t}_1 \ldots \models \mathtt{e}_n : \mathtt{t}_n) \Rightarrow (\models \mathscr{T}^n_{\mathcal{E}(\mathtt{e})}\ \mathtt{e}_1 \ldots \mathtt{e}_n : \mathtt{ut}^{\mathbf{n}})$$

We have $\models \mathtt{e} : \mathtt{t}_1 \to \cdots \to \mathtt{t}_n \to \mathtt{ut}^{\top}$. Now either

$$\vdash \mathtt{e}\ \mathtt{e}_1 \ldots \mathtt{e}_n \Downarrow \mathtt{v}$$

or

$$\vdash \mathtt{e}\ \mathtt{e}_1 \ldots \mathtt{e}_n \not\Downarrow$$

holds. In the first case we apply Lemma 22 and then have $\vdash \mathscr{T}^n_{\mathcal{E}(\mathtt{e})}\ \mathtt{e}_1 \ldots \mathtt{e}_n \Downarrow \mathtt{v}'$. In the second case we apply Lemma 23 and then have $\vdash \mathscr{T}^n_{\mathcal{E}(\mathtt{e})}\ \mathtt{e}_1 \ldots \mathtt{e}_n \Downarrow \mathtt{v}'$. In both cases we have $\vdash \mathscr{T}^n_{\mathcal{E}(\mathtt{e})} \mathtt{e}_1 \ldots \mathtt{e}_n \Downarrow \mathtt{v}'$ so it must be the case that $\models \mathscr{T}^n_{\mathcal{E}(\mathtt{e})} \mathtt{e}_1 \ldots \mathtt{e}_n : \mathtt{ut}^{\mathbf{n}}$ holds. □

We now lift the notion of semantic equivalence to conjunction types:

Lemma 33

$$((\models \mathtt{e}_1 : \mathtt{ct}) \wedge (\mathtt{e}_1 \sim_{\mathcal{E}(\mathtt{ct})} \mathtt{e}_2)) \Rightarrow (\models \mathtt{e}_2 : \mathtt{ct})$$

Proof. The lemma is shown by a straightforward induction in the type ct. In all cases we know that both $\emptyset \vdash \mathtt{e}_1 : \mathcal{E}(\mathtt{ct})$ and $\emptyset \vdash \mathtt{e}_2 : \mathcal{E}(\mathtt{ct})$ can be inferred. We refer to [17] for the full details. □

The Corollaries 34–37 below are all applications of Lemma 33: they are all proven by showing that the two terms are semantically equivalent and then applying Lemma 33.

Corollary 34

$$(\models (\lambda \mathtt{x}.\mathtt{e})\mathtt{e}' : \mathtt{ct}) \Leftrightarrow (\models \mathtt{e}[\mathtt{e}'/\mathtt{x}] : \mathtt{ct})$$

Corollary 35

$$(\models \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ (\mathtt{e}_2\ \mathtt{e}')\ \mathtt{else}\ (\mathtt{e}_3\ \mathtt{e}') : \mathtt{ct}) \Rightarrow$$

$$(\models (\mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3)\ \mathtt{e}' : \mathtt{ct})$$

Corollary 36

$$(\models \mathtt{e}(\mathtt{fix}_n\ \mathtt{e}):\mathtt{ct}) \Leftrightarrow (\models \mathtt{fix}_{n+1}\ \mathtt{e}:\mathtt{ct})$$

Corollary 37

$$(\models \mathtt{e}(\mathtt{fix}\ \mathtt{e}):\mathtt{ct}) \Leftrightarrow (\models \mathtt{fix}\ \mathtt{e}:\mathtt{ct})$$

Provided $\mathtt{e}_1$ has the type $\mathtt{Bool}^{\mathbf{n}}$ and both $\mathtt{e}_2$ and $\mathtt{e}_3$ are valid of the conjunction type ct, the conditional is valid of the type ct:

Fact 38

$$((\models \mathtt{e}_1:\mathtt{Bool}^{\mathbf{n}}) \wedge (\models \mathtt{e}_2:\mathtt{ct}) \wedge (\models \mathtt{e}_3:\mathtt{ct})) \Leftrightarrow$$
$$(\models \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3:\mathtt{ct})$$

Proof. The lemma is shown by induction in the conjunction type ct. In all cases we are using that $\mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3$ has the underlying type $\varepsilon(\mathtt{ct})$. □

Provided $\mathtt{e}_1$ has the type $\mathtt{Bool}^{\top}$ and both $\mathtt{e}_2$ and $\mathtt{e}_3$ are valid of the conjunction type ct, and ct can describe bottom, then the conditional is valid of the type ct:

Fact 39

$$((\models \mathtt{e}_1:\mathtt{Bool}^{\top}) \wedge (\models \mathtt{e}_2:\mathtt{ct}) \wedge (\models \mathtt{e}_3:\mathtt{ct}) \wedge \mathrm{BOT}(\mathtt{ct})) \Rightarrow$$
$$(\models \mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3:\mathtt{ct})$$

Proof. The lemma is shown by induction in the conjunction type ct. In all cases we are using that $\mathtt{if}\ \mathtt{e}_1\ \mathtt{then}\ \mathtt{e}_2\ \mathtt{else}\ \mathtt{e}_3$ has the underlying type $\varepsilon(\mathtt{ct})$. □

Next we show that our rules for $\leqslant_{\mathrm{ST}}$ and $\leqslant_{\mathrm{CT}}$ are sound:

Lemma 40 (Soundness of $\leqslant_{\mathrm{ST}}$)

$$((\models \mathtt{e}:\mathtt{t}_1) \wedge (\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2)) \Rightarrow (\models \mathtt{e}:\mathtt{t}_2)$$

Proof. We assume that $\models \mathtt{e}:\mathtt{t}_1$ is true and that $\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2$ can be inferred, then we show by induction in the proof-tree of $\mathtt{t}_1 \leqslant_{\mathrm{ST}} \mathtt{t}_2$ that $\models \mathtt{e}:\mathtt{t}_2$ is true. Throughout the proof we use Fact 28 to tell us that $\emptyset \vdash \mathtt{e}:\varepsilon(\mathtt{t}_2)$.

The only nontrivial case is the case [monotone]: We assume $(\models \mathtt{e} : \mathtt{t}_1 \to \mathtt{t}_2)$ which is equivalent to

$$\forall \mathtt{e}'' : (\models \mathtt{e}'' : \mathtt{t}_1) \Rightarrow (\models \mathtt{e}\ \mathtt{e}'' : \mathtt{t}_2)$$

We also assume that $\mathtt{t}_1 \to \mathtt{t}_2 \leqslant_{\mathrm{ST}} \downarrow \mathtt{t}_1 \to \downarrow \mathtt{t}_2$ can be inferred. Fact 29(a) gives $\mathtt{t}_2 \leqslant_{\mathrm{ST}} \downarrow \mathtt{t}_2$ and as we did not apply the rule [monotone] in the proof Fact 29(a) we can apply the induction hypothesis to get

$$\forall \mathtt{e}' : (\models \mathtt{e}' : \mathtt{t}_1) \Rightarrow (\models \mathtt{e}\ \mathtt{e}' : \downarrow \mathtt{t}_2)$$

We want to show

$$\forall \mathtt{e}' : (\models \mathtt{e}' : \downarrow \mathtt{t}_1) \Rightarrow (\models \mathtt{e}\ \mathtt{e}' : \downarrow \mathtt{t}_2)$$

and we do so by induction on $\mathtt{t}_2$. In case where $\mathtt{t}_2 = \mathtt{ut}_2^{\mathbf{b}}$ we need to do induction in $\mathtt{t}_1$ also. In the case where $\mathtt{t}_1 = \mathtt{ut}_1^{\mathbf{n}}$ we make use of the terminating terms: we know that

$$\forall \mathtt{e}' : \models \mathtt{e}' : \mathtt{ut}_1^{\mathbf{n}} \Rightarrow \models \mathtt{e}\ \mathtt{e}' : \mathtt{ut}_2^{\mathbf{b}}$$

and we want to show

$$\forall \mathtt{e}' : \models \mathtt{e}' : \mathtt{ut}_1^{\top} \Rightarrow \models \mathtt{e}\ \mathtt{e}' : \mathtt{ut}_2^{\mathbf{b}}$$

We do so by contradiction; so suppose that $\models \mathtt{e}' : \mathtt{ut}_1^{\top}$ but that $\not\models \mathtt{e}\ \mathtt{e}' : \mathtt{ut}_2^{\mathbf{b}}$ holds. This means that $\vdash \mathtt{e}\ \mathtt{e}' \Downarrow \mathtt{v}$ for some v. Hence from Lemma 24 we have $\vdash \mathtt{e}\mathscr{T}^0_{\varepsilon(\mathtt{e}')} \Downarrow \mathtt{v}'$ for some v'. But as $\models \mathscr{T}^0_{\varepsilon(\mathtt{e}')} : \mathtt{ut}_1^{\mathbf{n}}$ this contradicts $\models \mathtt{e}\ \mathtt{e}' : \mathtt{ut}_2^{\mathbf{b}}$ and we conclude that the claim must be true.

We refer to [17] for full details of this proof. □

Lemma 41 (Soundness of $\leqslant_{\mathrm{CT}}$)

$$((\models \mathtt{e} : \mathtt{ct}_1) \wedge (\mathtt{ct}_1 \leqslant_{\mathrm{CT}} \mathtt{ct}_2)) \Rightarrow (\models \mathtt{e} : \mathtt{ct}_2)$$

Proof. The lemma is shown by a straightforward induction in the proof-tree of $\mathtt{ct}_1 \leqslant_{\mathrm{CT}} \mathtt{ct}_2$. □

We know from the semantics that $(\mathtt{fix}_0\ \mathtt{e})$ cannot evaluate hence it is valid of any type that can describe non-termination:

Lemma 42

$$(\mathrm{BOT}(\mathtt{t}_1) \wedge \varepsilon(\mathtt{t}_1) = \varepsilon(\mathtt{t}_2) \wedge \models \mathtt{e} : \mathtt{t}_1 \to \mathtt{t}_2) \Rightarrow (\models \mathtt{fix}_0\ \mathtt{e} : \mathtt{t}_1)$$

Proof. It is easy to show that $\models \mathtt{fix}_0\ \mathtt{e} : \varepsilon(\mathtt{t}_1)^{\mathbf{b}}$ holds. Since we have shown that $\mathrm{BOT}(\mathtt{t}_1)$ implies $\varepsilon(\mathtt{t}_1)^{\mathbf{b}} \leqslant_{\mathrm{ST}} \mathtt{t}_1$ (Lemma 30) we obtain the result using Lemma 41. □

Motivated by Fig. 5 we may clarify the relationship between $\mathtt{fix}_j$ and $\mathtt{fix}$ as follows:

Lemma 43

$$(\exists j_0, j_1 : \forall k \geqslant 0 : (\models \mathtt{fix}_{j_0+j_1\times k}\ \mathtt{e} : \mathtt{t})) \Rightarrow (\models \mathtt{fix}\ \mathtt{e} : \mathtt{t})$$

provided e is without any $\mathtt{fix}_j$

Proof. The lemma is proved by induction on the strictness and totality type t. For the full details see [17]. □

4.3. The soundness proof

Finally we can prove Theorem 26:

Theorem 26 (Soundness). *For expressions* e *without any* $\mathtt{fix}_n$ *and* $\mathscr{T}^n$ *we have*

$$(\overline{\mathtt{x}} : \overline{\mathtt{t}} \vdash_{\mathrm{ST}} \mathtt{e} : \mathtt{ct}) \Rightarrow (\forall \overline{\mathtt{v}} : (\models \overline{\mathtt{v}} : \overline{\mathtt{t}}) \Rightarrow (\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{ct}))$$

Proof. We assume that $\mathrm{A} \vdash_{\mathrm{ST}} \mathtt{e} : \mathtt{ct}$ and that $(\models \overline{\mathtt{v}} : \overline{\mathtt{t}})$ is true, then we prove by induction in the proof-tree for $\mathrm{A} \vdash_{\mathrm{ST}} \mathtt{e} : \mathtt{ct}$ that $\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{ct}$ is true. The proof is rather straightforward and we illustrate one typical case below and refer to [17] for the full details.

The case [fix]: We assume $\mathrm{A} \vdash_{\mathrm{ST}} \mathtt{fix}\ \mathtt{e} : \mathtt{t}_n$, $\mathrm{BOT}(\mathtt{t}_1)$, $\mathtt{t}_q \leqslant_{\mathrm{ST}} \mathtt{t}_p$, $p < q$, and that $\models \overline{\mathtt{v}} : \overline{\mathtt{t}}$ is true. From the [fix]-rule we get

$$\mathrm{A} \vdash_{\mathrm{ST}} \mathtt{e} : \mathtt{t}_1 \to \mathtt{t}_2 \wedge \mathtt{t}_2 \to \mathtt{t}_3 \wedge \cdots \wedge \mathtt{t}_{n-1} \to \mathtt{t}_n$$

By applying the induction hypothesis we get

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_1 \to \mathtt{t}_2 \wedge \mathtt{t}_2 \to \mathtt{t}_3 \wedge \cdots \wedge \mathtt{t}_{n-1} \to \mathtt{t}_n$$

which is equivalent to

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_1 \to \mathtt{t}_2$$

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_2 \to \mathtt{t}_3$$

$$\vdots$$

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_{n-1} \to \mathtt{t}_n$$

From Lemma 42 we have $\models \mathtt{fix}_0\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_1$. By applying

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_1 \to \mathtt{t}_2$$

we get $\models \mathtt{e}(\mathtt{fix}_0\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]) : \mathtt{t}_2$ and Fact 36 gives $\models \mathtt{fix}_1\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_2$. Now by applying $\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}] : \mathtt{t}_2 \to \mathtt{t}_3$ again we get $\models \mathtt{e}(\mathtt{fix}_1\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]) : \mathtt{t}_3$ and Fact 36

gives $\models \mathtt{fix}_2\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_3$. We arrive at $\models \mathtt{fix}_{q-1}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_q$. Now since we have $\mathtt{t}_q \leqslant_{\mathrm{ST}} \mathtt{t}_p$ we can apply Lemma 40 to get $\models \mathtt{fix}_{q-1}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_p$. By applying $\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_p \to \mathtt{t}_{p+1}$ we have $\models \mathtt{e}(\mathtt{fix}_{q-1}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]):\mathtt{t}_{p+1}$ and using Fact 36 we get $\models \mathtt{fix}_{q-1+1}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_{p+1}$. In this way we arrive at

$$\forall k \geqslant 0: \models \mathtt{fix}_{q-1+(q-p)k}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_q$$

and using Lemma 43 this gives $\models \mathtt{fix}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_q$. Applying

$$\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_q \to \mathtt{t}_{q+1}$$

gives $\models \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]\ (\mathtt{fix}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]):\mathtt{t}_{q+1}$. Now Fact 37 gives $\models \mathtt{fix}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_{q+1}$. In this way we arrive at $\models \mathtt{fix}\ \mathtt{e}[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_n$ that is $\models (\mathtt{fix}\ \mathtt{e})[\overline{\mathtt{v}}/\overline{\mathtt{x}}]:\mathtt{t}_n$ as required. □

5. Conclusion

We have described an inference system for combining strictness and totality analysis and we have proved the analysis sound with respect to a natural-style operational semantics.

We have briefly compared the results obtained by our analysis to those obtained by e.g. [6, 2, 8, 9, 22]. These systems are incomparable to ours in that in some cases we get more precise results, in others they do. One may note that the type systems of Jensen [6] and Benton [2] allow general conjunction types; in our view the reason why this causes no problems is that it is not possible to construct empty types because their type system excludes the **n** annotation.

The totality analysis is rather weak. Consider the factorial function:

$$\mathtt{fac} ::= \mathtt{fix}\ (\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{if}\ =\ \mathtt{x}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ *(\mathtt{f}\ (-\ \mathtt{x}\ 1))\mathtt{x})$$

We can infer the strictness and totality type for the factorial function

$$(\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\top}) \wedge (\mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\mathbf{b}})$$

but not the type $\mathtt{Int}^{\mathbf{n}} \to \mathtt{Int}^{\mathbf{n}}$. In order to do that we have to define a well-founded ordering as done in [14].

An open problem is the meaningful integration of data-types like lists. For the strictness part one may be inspired by [20]. Consider the type `B list` where `B` is a base-type. The strictness and totality type $(\mathtt{B}^{\mathbf{n}})\mathtt{list}$ might then describe the finite lists with no bottom elements, the type $(\mathtt{B}^{\mathbf{b}})\mathtt{list}$ might describe the infinite lists or lists with bottom elements, and the strictness and totality type $(\mathtt{B}^{\top})\mathtt{list}$ might describe all list. A strictness and totality type of the `map` function would then be

$$(\mathtt{B}^{\mathbf{n}} \to \mathtt{B}'^{\mathbf{n}}) \to (\mathtt{B}^{\mathbf{n}})\mathtt{list} \to (\mathtt{B}'^{\mathbf{n}})\mathtt{list}$$

Similarly, `foldl` and `foldr` will have strictness and totality types

$$(\mathtt{B}^{\mathbf{n}} \to \mathtt{B}'^{\mathbf{n}} \to \mathtt{B}^{\mathbf{n}}) \to \mathtt{B}^{\mathbf{n}} \to (\mathtt{B}'^{\mathbf{n}})\mathtt{list} \to \mathtt{B}^{\mathbf{n}}$$

and

$$(\mathtt{B}^{\mathbf{n}} \to \mathtt{B}'^{\mathbf{n}} \to \mathtt{B}'^{\mathbf{n}}) \to \mathtt{B}'^{\mathbf{n}} \to (\mathtt{B}^{\mathbf{n}})\mathtt{list} \to \mathtt{B}'^{\mathbf{n}}$$

respectively. However, to get this information from the analysis we need to analyse fixpoints in a better way, e.g. as suggested in [14].

In [18, 17] we have lifted the restriction on the placement of conjunction; this results in a somewhat more powerful system. However, we were unable to prove the soundness of that system using operational semantics and had to resort to denotational semantics.

Acknowledgements

Thanks to LOMAPS (Esprit Basic Research) and DART (Danish Science Research Council) for partial funding.

Appendix A. The power of the fix-rules

Previous work on strictness analysis [6, 8, 2, 9] contain only a simple fix-rule corresponding to our [fix1]-rule rather than our more general [fix]-rule. In this section we will investigate the extent to which this is essential.

Let $\mathscr{A}$ be a set of permissible annotations; so far we used $\mathscr{A} = \{\mathbf{n}, \mathbf{b}, \top\}$ but we shall consider also the restriction $\mathscr{A} = \{\mathbf{b}, \top\}$ that disallows $\mathbf{n}$ and that corresponds more closely to the aims of [6, 8, 2, 9]. Note that the side-condition BOT(t) is trivially true when $\mathscr{A} = \{\mathbf{b}, \top\}$. Let $\vdash^{\mathscr{A}}_{\mathrm{fix}}$ be the inference system of Fig. 4 but with annotations in $\mathscr{A}$. Similarly let $\vdash^{\mathscr{A}}_{\mathrm{fix1}}$ be the system where [fix] is replaced by [fix1] and let $\vdash^{\mathscr{A}}_{\mathrm{fix2}}$ be the system where [fix] is replaced by [fix2]. Note that $\vdash^{\{\mathbf{b},\top\}}_{\mathrm{fix1}}$ is the system of [9].

For any two inference systems $\vdash^{\mathscr{A}_1}_{\phi_1}$ and $\vdash^{\mathscr{A}_2}_{\phi_2}$ write

$$\vdash^{\mathscr{A}_1}_{\phi_1} \subseteq \vdash^{\mathscr{A}_2}_{\phi_2} \quad \text{for } \mathrm{A} \vdash^{\mathscr{A}_1}_{\phi_1} \mathtt{e} : \mathtt{t} \Rightarrow \mathrm{A} \vdash^{\mathscr{A}_2}_{\phi_2} \mathtt{e} : \mathtt{t}$$

and

$$\vdash^{\mathscr{A}_1}_{\phi_1} = \vdash^{\mathscr{A}_2}_{\phi_2} \quad \text{for } \vdash^{\mathscr{A}_1}_{\phi_1} \subseteq \vdash^{\mathscr{A}_2}_{\phi_2} \wedge \vdash^{\mathscr{A}_2}_{\phi_2} \subseteq \vdash^{\mathscr{A}_1}_{\phi_1}$$

$$\vdash^{\mathscr{A}_1}_{\phi_1} \subset \vdash^{\mathscr{A}_2}_{\phi_2} \quad \text{for } \vdash^{\mathscr{A}_1}_{\phi_1} \subseteq \vdash^{\mathscr{A}_2}_{\phi_2} \wedge \neg(\vdash^{\mathscr{A}_2}_{\phi_2} \subseteq \vdash^{\mathscr{A}_1}_{\phi_1})$$

It is immediate that

$$\vdash^{\mathscr{A}}_{\mathrm{fix1}} \subseteq \vdash^{\mathscr{A}}_{\mathrm{fix2}} \subseteq \vdash^{\mathscr{A}}_{\mathrm{fix}}$$

and that

$$\vdash^{\{\mathbf{b},\top\}}_{\phi} \subseteq \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\phi}$$

for all $\mathscr{A}$ and $\phi \in \{\mathrm{fix}, \mathrm{fix1}, \mathrm{fix2}\}$. We now consider the extent to which the inclusions are proper or are equalities; the results are summarised in Table 1.

Table 1
Relation between the fix-rules

Annotations $\mathscr{A}$	Fix-rules
$\{\mathbf{b}, \top\}$	$\vdash^{\mathscr{A}}_{\text{fix1}} = \vdash^{\mathscr{A}}_{\text{fix2}}$ $\vdash^{\mathscr{A}}_{\text{fix2}} \subset \vdash^{\mathscr{A}}_{\text{fix}}$
$\{\mathbf{n}, \mathbf{b}, \top\}$	$\vdash^{\mathscr{A}}_{\text{fix1}} \subset \vdash^{\mathscr{A}}_{\text{fix2}}$ $\vdash^{\mathscr{A}}_{\text{fix2}} \subset \vdash^{\mathscr{A}}_{\text{fix}}$

Claim. $\vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} = \vdash^{\{\mathbf{b},\top\}}_{\text{fix2}}$

In order to show that $\vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} = \vdash^{\{\mathbf{b},\top\}}_{\text{fix2}}$ it suffices to show that the rule [fix2] can be derived from the rule [fix1]. For this assume

$$A \vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} \mathtt{e} : t_1 \to t_2$$

$$t_2 \leqslant_{ST} t_1$$

$$\text{BOT}(t_1)$$

so that

$$A \vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} \mathtt{fix\ e} : t_2$$

can be inferred. Since none of the types involves the annotation $\mathbf{n}$ it must be the case that $\text{BOT}(\mathtt{t}) = \mathtt{tt}$ for all types $\mathtt{t}$. We can now construct the proof-tree

$$\dfrac{\dfrac{A \vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} \mathtt{e} : t_1 \to t_2 \qquad \dfrac{t_2 \leqslant_{ST} t_1}{t_1 \to t_2 \leqslant_{ST} t_2 \to t_2}}{A \vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} \mathtt{e} : t_2 \to t_2} \qquad \text{BOT}(t_2)}{A \vdash^{\{\mathbf{b},\top\}}_{\text{fix1}} \mathtt{fix\ e} : t_2}\ [\text{fix1}]$$

and this proves our claim.

Claim. $\vdash^{\{\mathbf{b},\top\}}_{\text{fix2}} \subset \vdash^{\{\mathbf{b},\top\}}_{\text{fix}}$

To verify that $\vdash^{\{\mathbf{b},\top\}}_{\text{fix2}} \subset \vdash^{\{\mathbf{b},\top\}}_{\text{fix}}$ we must show that there exists a term $\mathtt{e}$ and a type $\mathtt{t}$ and an assumption list A, such that $A \vdash^{\{\mathbf{b},\top\}}_{\text{fix}} \mathtt{e} : \mathtt{t}$ can be inferred and we cannot infer $A \vdash^{\{\mathbf{b},\top\}}_{\text{fix2}} \mathtt{e} : \mathtt{t}$.
For this we take

$$\mathtt{e} = \mathtt{fix}\ (\lambda \mathtt{f}.\lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{if}\ (=\ 0\ \mathtt{z})\ \mathtt{then}\ +\ \mathtt{x}\ \mathtt{y}\ \mathtt{else}\ \mathtt{f}\ \mathtt{y}\ \mathtt{x}\ (-\ \mathtt{z}\ 1))$$

$$t_1 = \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$

$$t_2 = \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}} \to \mathtt{Int}^{\top} \to \mathtt{Int}^{\mathbf{b}}$$

In Example 8 we have shown how to infer:

$$\emptyset \vdash^{\{\mathbf{b},\top\}}_{\text{fix}} \texttt{fix e} : \texttt{t}_1 \wedge \texttt{t}_2$$

and we argued about the unlikeliness of being able to infer $\emptyset \vdash^{\{\mathbf{b},\top\}}_{\text{fix2}} \texttt{fix e} : \texttt{t}_1 \wedge \texttt{t}_2$ (as is indeed stated also in [9]).

Claim. $\vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix1}} \subset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}}$

When we go to the $\{\mathbf{n},\mathbf{b},\top\}$-part (both strictness and totality information on the types) the two rules [fix1] and [fix2] are no longer equivalent. Consider the term $\texttt{fix}\ (\lambda \texttt{x}.7)$ and the type $\texttt{Int}^{\mathbf{n}}$. We can infer

$$\emptyset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}} \lambda \texttt{x}.7 : \texttt{Int}^{\mathbf{n}} \to \texttt{Int}^{\mathbf{n}}$$

but this does not suffice for using the rule [fix1] to infer

$$\emptyset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}} \texttt{fix}\ (\lambda \texttt{x}.7) : \texttt{Int}^{\mathbf{n}}$$

because BOT($\texttt{Int}^{\mathbf{n}}$) fails. However we can infer

$$\emptyset \vdash_{\text{ST}} \lambda \texttt{x}.7 : \texttt{Int}^{\top} \to \texttt{Int}^{\mathbf{n}}$$

and we can then apply the rule [fix2] to get the desired type. This argument shows

$$\neg(\text{A} \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}} \texttt{e} : \texttt{t} \Rightarrow \text{A} \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix1}} \texttt{e} : \texttt{t})$$

and thereby we have $\vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix1}} \subset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}}$.

Claim. $\vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}} \subset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix}}$

To argue that $\vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix2}} \subset \vdash^{\{\mathbf{n},\mathbf{b},\top\}}_{\text{fix}}$ when we consider the full strictness and totality analysis we can use the same term and type as for showing $\vdash^{\{\mathbf{b},\top\}}_{\text{fix2}} \subset \vdash^{\{\mathbf{b},\top\}}_{\text{fix}}$.

References

[1] S. Abramsky, Abstract interpretation, logical relations and Kan extensions, J. Logic Comput. 1 (1) (1990) 5–39.

[2] N. Benton, Strictness analysis of functional programs, Ph.D. Thesis, University of Cambridge, 1993. Technical Report No. 309, University of Cambridge, 1993.

[3] G.L. Burn, C. Hankin, S. Abramsky, Strictness analysis for higher-order functions, Science of Computer Programming 7 (3) (1986) 249–278.

[4] P. Cousot, R. Cousot, Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages), invited paper, in: Proc. 1994 Internat. Conf. on Computer Languages, ICCL'94, IEEE Computer Society Press, 1994, pp. 95–112. Available by WWW from lix.polytechnique.fr file://lix.polytechnique.fr/pub/ESPRIT/LOMAPS/LOMAPS-ENS-X-1.ps.gz.

[5] J. Despeyroux, Proof of translation in natural semantics, in: Proc. Symposium on Logic in Computer Science (1986).
[6] T.P. Jensen, Strictness analysis in logical form, in: Proc. FPCA'91, Lecture Notes in Computer Science, vol. 523, Springer, Berlin, 1991, pp. 352–366.
[7] T.P. Jensen, Abstract interpretation in logical form, Ph.D. Thesis, University of London, Imperial College, 1992.
[8] T.P. Jensen, Disjunctive strictness analysis, in: Proc. LICS'92, 1992, pp. 174–185.
[9] T.-M. Kuo, P. Mishra, Strictness analysis: A new perspective based on type inference, in: Proc. FPCA'89, ACM Press, New York, 1989, pp. 260–272.
[10] R. Milner, A theory of type polymorphism in programming, J. Comput. System Sci. 17 (1978) 348–375.
[11] A. Mycroft, The theory and practice of transforming call-by-need into call-by-value, in: Proc. 4th Internat. Symp. on Programming, Lecture Notes in Computer Science, vol. 83, Springer, Berlin, 1980, pp. 269–281.
[12] A. Mycroft, Abstract interpretation and optimising transformation for applicative programs, Ph.D. Thesis, University of Edinburgh, Scotland, 1981.
[13] A. Mycroft, F. Nielson, Strong abstract interpretation using power domain (extended abstract), in: Proc. ICALP'83, Lecture Notes in Computer Science, vol. 154, Springer, Berlin, 1983.
[14] F. Nielson, H.R. Nielson, Termination analysis based on operational semantics, Technical Report DAIMI PB-492, Aarhus University, 1995.
[15] G.D. Plotkin, LCF considered as a programming language, Theoret. Comput. Sci. 5 (1977) 223–255.
[16] G.D. Plotkin, A structural approach to operational semantics, Technical Report, Aarhus University, 1981. DAIMI FN-19.
[17] K.L. Solberg, Annotated type systems for program analysis, Ph.D. Thesis, Odense University, Denmark, 1995, Technical Report DAIMI PB-498, Aarhus University, Denmark.
[18] K.L. Solberg, Strictness and totality analysis with conjunction, in: Proc. TAPSOFT'95, Lecture Notes in Computer Science, vol. 915, Springer, Berlin, 1995, pp. 501–515.
[19] D.A. Turner, Miranda: A non-strict functional language with polymorphic types, in: Proc. FPCA'85, Lecture Notes in Computer Science, vol. 201, Springer, Berlin, 1985, pp. 1–16.
[20] P. Wadler, Strictness analysis on non-flat domains (by abstract interpretation over finite domains, in: S. Abramsky, C. Hankin (Eds.), Abstract Interpretation of Declarative Languages, Ellis Horwood, Chichester, UK, 1987, pp. 266–275.
[21] P. Wadler, J. Hughes, Projections for strictness analysis, in: Proc. FPCA'87, Lecture Notes in Computer Science, vol. 27, Springer, Berlin, 1987.
[22] D.A. Wright, A new technique for strictness analysis, in: Proc. TAPSOFT'91, Lecture Notes in Computer Science, vol. 494, Springer, Berlin, 1991, pp. 260–272.