# Polymorphic Subtyping for Side Effects

Torben Amtoft & Flemming Nielson & Hanne Riis Nielson
Computer Science Department, Aarhus University, Denmark
e-mail:{tamtoft,fn,hrn}@daimi.aau.dk

October 9, 1997

**Abstract**

The integration of polymorphism (in the style of the ML `let`-construct), subtyping, and effects (modelling assignment or communication) into one common type system has proved remarkably difficult. This paper presents a type system for (a core subset of) Concurrent ML that extends the ML type system in a conservative way and that employs all these features; and in addition causality information has been incorporated into the effects (which may therefore be termed "behaviours").

The semantic soundness of the system is established via a subject reduction result. An inference algorithm is presented; it is proved sound and (in a certain sense) also complete. A prototype system based on this algorithm has been implemented and can be experienced on the WWW; thanks to a special post-processing phase it produces quite readable and informative output.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

The last decade has seen a number of papers addressing the difficult task of developing type systems for languages that admit polymorphism in the style of the ML `let`-construct, that admit subtyping, and that admit effects as may arise from assignment or communication.

This is a problem of practical importance. The programming language Standard ML has been joined by a number of other high-level languages demonstrating the power of polymorphism for large scale software development. Already Standard ML contains imperative effects in the form of `ref`-types that can be used for assignment; closely related languages like Concurrent ML or Facile further admit primitives for synchronous communication. Finally, the trend towards integrating aspects of object orientation into these languages necessitates a study of subtyping.

Apart from the need to type such languages we see a need for type systems integrating polymorphism, subtyping, and effects in order to be able to continue the present development of annotated type and effect systems for a number of static program analyses; example analyses include control flow analysis, binding time analysis and communication analysis. This will facilitate modular proofs of correctness while at the same time allowing the inference algorithms to generate syntax-free constraints that can be solved efficiently.

## 1.2 State of the Art

**Polymorphism.** One of the pioneering papers in the area is [11] that developed the first polymorphic type inference system, and an algorithm, for the applicative fragment of ML; a shorter presentation for the typed $\lambda$-calculus with `let` is given in [4].

**Subtyping.** Since then many papers have studied how to integrate subtyping. A number of early papers did so by mainly focusing on the typed $\lambda$-calculus and only briefly dealing with `let` [12, 6]. Later papers have treated polymorphism in full generality [26, 8]. A key ingredient in these approaches is the simplification of the enormous set of constraints into something manageable [5, 26].

**Effects.** Already ML necessitates an incorporation of imperative effects due to the presence of `ref`-types. A pioneering paper in the area is [30] that develops a distinction between imperative and applicative type variables: for *creation* of a reference cell we demand that its type contain imperative variables only; and one is not allowed to generalise over imperative variables unless the expression in question is *non-expansive* (i.e. does not expand the store) which will be the case if it is an identifier or a function abstraction.

The problem of typing ML with references (but without subtyping) has lead to a number of attempts to improve upon [30]; this includes the following:

- [32] is similar in spirit to [30] in that one is not allowed to generalise over a type variable if a reference cell has been *created* with a type containing this variable; to trace such variables the type system is augmented with *effects*. Effects may be approximated by larger effects, that is the system employs *subeffecting*.
- [28] can be considered a refinement of [32] in that effects also record the *region* in which a reference cell is created or a read/write operation is performed; this information enables one to "mask" effects which have taken place in "inaccessible" regions.
- [10] presents a somewhat alternative view: here focus is not on detecting *creation* of reference cells but rather to detect their *use*; this means

that if an identifier occurs free in a function closure then all variables in its type have to be "examined". This method is quite powerful but unfortunately it fails to be a conservative extension of ML: some purely applicative programs which are typeable in ML may be untypeable in this system.

The surveys in [28, section 11] and in [32, section 5] show that many of these (and other) systems are incomparable, in the sense that for any two approaches it will often be the case that there are programs which are accepted by one of them but not by the other, and vice versa. Our approach (which will be illustrated by a fragment of Concurrent ML but is equally applicable to Standard ML with references) involves subtyping which is strictly more powerful than subeffecting (as shown in Sect. 2.5); apart from this we do not attempt to measure its strength relative to other approaches but we do demonstrate that it is a conservative extension of ML (Sect. 2.8).

**Integration.** In the area of static program analysis, annotated type and effect systems have been used as the basis for control flow analysis [29] and binding time analysis [16, 7]. These papers typically make use of a polymorphic type system with subtyping and no effects, or a non-polymorphic type system with effects and subtyping. A more ambitious analysis is the approach of [17] to let annotated type and effect systems extract terms of a process algebra from programs with communication; this involves polymorphism and subeffecting but (presumably because the inference system is expressed without using constraints) the algorithmic issues are non-trivial [14]; [1] presents an algorithm that is sound as well as complete, but which generates constraints that we do not know how to solve in the general case. Finally we should mention [31] where effects are incorporated into ML types in order to deal with *region inference*.

The type system presented in [19] is a major step towards integrating polymorphism, subtyping, and effects; it generalises the subeffecting approach of [28] and admits effects into the subtyping approaches of [26, 8]. A key insight is that in order to establish semantic soundness (as is formally done in [2]) one must be very careful when deciding the set of variables over which to generalise in the inference rule for `let`: not only should this set be disjoint from the set of variables occurring in the effect (as is standard in effect systems, e.g. [28]) but it should also be *upwards closed* with respect to a constraint set. To keep the development in [19] as simple as possible, region information is omitted from the effects.

## 1.3 Major Achievement I: Causality

Chapter 2 reintroduces regions and further improves on [19] in that causality is incorporated into the effects, thus following [17], and we shall therefore prefer to use the word "behaviours" rather than "effects". At the same time we slightly reformulate the notion of upwards closure used in the generalisation rule (cf. the preceding paragraph). Judgements take the form

$$C, A \vdash e : t \,\&\, b$$

with $e$ an expression, $b$ a behaviour, $t$ a type annotated with behaviour information (as e.g. the function type $\texttt{int} \;\to^{b}\; \texttt{int}$), $C$ a set of constraints among types and behaviours and regions, and $A$ an environment. A subtyping relation is defined using a subeffecting relation on behaviours, with the usual contravariant ordering for function space.

## 1.4 Semantic Soundness

Chapter 3 addresses the soundness of the static semantics (i.e. the type system) wrt. a dynamic semantics. Statements of semantic soundness typically contain as premise that the inference system assigns a type $t$ to $e$ but the conclusion depends on the kind of dynamic semantics used: for a denotational semantics one may require (as in [11]) that the denotation of $e$ "has type" $t$; for a big-step (natural) semantics one may require (as in [30, 10]) that if $e \to v$ then $v$ "has type" $t$; for a small-step semantics [21] one requires (as in [33]) the following *subject reduction* property: if $e \to e'$ then the inference system also assigns $e'$ the type $t$. In addition, in order to ensure that "well-typed programs do not go wrong" [11] one must establish that "error configurations" (those which are "stuck") cannot be typed.

We shall choose a small-step semantics as we consider this the most appropriate for concurrent languages; the configurations of the transition system will be process pools $PP$ which map process identifiers into expressions. To get a flavour of how subject reduction is formulated in our setting consider the case where $PP$ rewrites to $PP'$ because process $p$ allocates a fresh channel $ch$ in region $\rho$ which is able to transmit values of type $t'$, and suppose that

$$C, A \vdash PP(p) : t \,\&\, b$$

holds: then Theorem 3.28 tells us that we also have

$$C, A[ch : t' \; \texttt{chan} \; \rho] \vdash PP'(p) : t \,\&\, b'$$

where $b$ approximates $t'$ CHAN $\rho; b'$ (that is, the sequential composition of the "current action" $t'$ CHAN $\rho$ and the "future action" $b'$). The general picture is much as in [17] that types are unchanged whereas the behaviours get "smaller" and the environments are "extended".

Extending the environment is a potential danger to semantic soundness, cf. the considerations in [30, section 5] where it was concluded that store operations in Standard ML are harmless unless they actually expand the store. In Example 2.6 it is demonstrated that channel allocations (the way our setting "expands the store") may be harmful unless one is very careful when deciding the set of variables over which to generalise in the rule for `let` in the inference system; the proof of Lemma 3.24 highlights how the judicious choice of generalisation strategy actually allows to extend the environment.

## 1.5 An Inference Algorithm

In Chap. 4 we shall aim at constructing a type reconstruction algorithm in the spirit of Milner's algorithm $\mathcal{W}$ [11]: given an expression $e$ and an environment $A$, the recursively defined function $\mathcal{W}$ will produce a substitution $S$, a type $t$, and a behaviour $b$. The definition in [11] employs unification [23]: if $e_1$ has been given type $t_0 \rightarrow t_1$ and $e_2$ has been given type $t_2$ then in order to type $e_1\, e_2$ one must unify $t_0$ and $t_2$. Unification works by decomposition: in order to unify $t_1 \rightarrow t_2$ and $t'_1 \rightarrow t'_2$ one recursively unifies $t_1$ with $t'_1$ and $t_2$ with $t'_2$. Decomposition is valid because types constitute a "free algebra": two types are equal if and only if they have the same top-level constructor and also their subcomponents are equal. However, this will not be the case for behaviours, and therefore $\mathcal{W}$ of [11] cannot immediately be generalised to work on annotated types.

We thus have to rethink the unification algorithm; and as the behaviours of this paper do not seem to satisfy simple algebraic properties (such as associativity or commutativity) it appears unlikely that we can adapt results from unification theory [24] (to get a unification algorithm producing a set of unifiers from which all other unifiers can be derived). Therefore we shall instead follow [9] and generate *behaviour constraints*: that is, in the process of unifing $t_1 \rightarrow^b t_2$ and $t'_1 \rightarrow^{b'} t'_2$ we generate constraints relating $b$ and $b'$.

In order to incorporate subtyping we also need to generate *type constraints* as in [6, 26]. The presence of type constraints is a consequence of our overall design: types and behaviours should be inferred simultaneously "from scratch", as is done by the algorithm $\mathcal{W}$ presented in Sect. 4.1. This should be compared with the approach in [29, chapter 5] where an effect system with subtyping but without polymorphism is presented; as the "underlying" types are given in advance it is sufficient to generate behaviour constraints.

The constraints generated by $\mathcal{W}$ have to be massaged so as to satisfy certain invariants and for this we devise the algorithm $\mathcal{F}$ (Sect. 4.3), inspired by [6]. Still the algorithm will produce a rather unwieldy number of constraints; to reduce this number substantially we may apply an algorithm $\mathcal{R}$ (defined in Sect. 4.4) which adapts the techniques of [5, 26].

## 1.6 Syntactic Soundness

In Sect. 4.5 we shall prove that $\mathcal{W}$ is (syntactically) sound, that is if $\mathcal{W}(A, e) = (S, t, b, C)$ then $C, S\,A \vdash e : t \,\&\, b$.

As the main distinguishing feature of our inference system (as mentioned above), essential for semantic soundness, was the choice of generalisation rule; so the distinguishing feature of our algorithm, essential for syntactic soundness (and eventually for syntactic completeness), is the choice of generalisation rule. This involves (rather similar to [32]) taking downwards closure of a set of variables with respect to a constraint set.

## 1.7 Major Achievement II: Completeness

Chapter 5 is devoted to the difficult task of proving the *completeness* of the algorithm presented in Chap. 4. Theorem 5.18 demonstrates that if

$$C^*, A^* \vdash e : \sigma^* \,\&\, b^*$$

and if certain well-formedness criteria are fulfilled (to be discussed in Sect. 5.2), then this judgement will be an "instance" of what is produced by $\mathcal{W}$.

## 1.8 Implementation

The resulting algorithm $\mathcal{W}$ (which employs $\mathcal{F}$ and $\mathcal{R}$) has been used as the basis of a prototype implementation, available for experimentation on the WWW[1]; we do not attempt to estimate the complexity of the algorithm. The system post-processes the constraints generated by $\mathcal{W}$ so as to produce readable output; in Chapter 6 we mention a selection of the techniques used and show that the resulting constraint set is in a certain sense bisimilar to the original constraints.

[3] contains a description of the system, illustrated by several examples, as well as a brief account of the underlying theory (to be developed in the rest of this document). It turns out [15] that the system greatly assists in validating a number of safety properties for "realistic" concurrent systems.

## 1.9 Future Work

We have seen that the present development integrates many features from previous approaches in the literature; below we mention some features that are *not* yet covered:

- unlike [6, 26] we do not allow inclusion between base types, such as `int` $\subseteq$ `real`;

[1] `http://www.daimi.aau.dk/~bra8130/TBA/TBA.html`.

- unlike [7, 31] we do not enable polymorphic recursion in the type annotations.

# Chapter 2

# The Static Semantics

For illustrating our approach we have chosen a variant of Concurrent ML (CML) [22, 20] which includes

- identifiers $x$, function abstractions `fn` $x \Rightarrow e$ and function applications $e_1\ e_2$ (as in the $\lambda$-calculus);
- polymorphic `let`-expressions (as in ML [11]);
- recursive functions and conditionals (to facilitate programming);
- constructors (for building data structures);
- base functions (for inspecting and decomposing data structures).

*Base functions* as well as *constructors* are divided into two classes: the *sequential* (known from ML) and the *non-sequential* (incorporating the concurrency aspect); with $F$ ranging over base functions, all unary, and with $C^n$ ranging over $n$-ary constructors ($n \geq 0$) we thus have

$$
\begin{array}{lcl}
F & ::= & F_s \mid F_c \\
C^n & ::= & C_s^n \mid C_c^n
\end{array}
$$

The sequential constructors will at least include the unique element of the unit type, the two booleans, numbers ($n \in Num$), `pair` for constructing pairs, and `nil` and `cons` for constructing lists:

$$C_s^0 ::= \texttt{()} \mid \texttt{true} \mid \texttt{false} \mid n \mid \texttt{nil}$$
$$C_s^2 ::= \texttt{pair} \mid \texttt{cons}$$

The sequential base functions will at least include a selection of arithmetic operations, `fst` and `snd` for decomposing a pair, and `hd`, `tl` and `null` for decomposing and inspecting a list:

$$\begin{array}{lcl} F_s & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{=} \mid \cdots \\ & \mid & \texttt{fst} \mid \texttt{snd} \mid \texttt{hd} \mid \texttt{tl} \mid \texttt{null} \end{array}$$

The unique flavour of Concurrent ML is due to the non-sequential constants which are the primitives for communication; we include five of these but more (in particular `choose` and `wrap`) can be added.

$$C_c^1 ::= \texttt{transmit} \mid \texttt{receive}$$
$$F_c ::= \texttt{sync} \mid \texttt{channel}^l \mid \texttt{spawn}$$

The non-sequential constructors are `transmit` and `receive`: rather than actually enabling a communication they create *delayed communications* which are first-class entities that can be passed around freely. This leads to a very powerful programming discipline, in particular in the presence of `choose` and `wrap`[1], as is discussed in [22]. The non-sequential base functions are `spawn`, `sync`, $\texttt{channel}^l$ and these are explained below.

The function `spawn` spawns a new process $e$ when applied to the expression `fn` $x \Rightarrow e$ (where $x$ is not used in $e$); this process will then execute concurrently with the other processes, one of which is the program itself.

The function `sync` synchronises (i.e. activates) a delayed communication. Thus one process can send the value of $e$ to another process by the expression[2] `sync (transmit (ch,`$e$`))` where communication takes place along the channel `ch`. Similarly a process can receive a value from another process by the expression[3] `sync (receive (ch))`.

A function $\texttt{channel}^l$ allocates a new typed communication channel when applied to `()`; in order to keep track of the origin of the allocated channels

[1] To add these constants requires a non-trivial reformulation of the semantics presented in Chap. 3.

[2] In CML, this can also be written `send (ch,`$e$`)`.

[3] In CML, this can also be written `accept (ch)`.

$$
\begin{array}{rcl}
e & ::= & x \mid \texttt{fn}\ x \Rightarrow e \mid e_1\ e_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \\
 & \mid & \texttt{rec}\ f\ x \Rightarrow e \mid \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
 & \mid & F < e > \mid C^n < e_1, \cdots, e_n >
\end{array}
$$

Figure 2.1: Expressions $e \in \mathit{Exp}$

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \texttt{fn}\ x \Rightarrow e \mid e_1\ e_2 \mid e_0 \mathbin{@}^s_n < e_1, \cdots, e_n > \\
 & \mid & \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid \texttt{rec}\ f\ x \Rightarrow e \mid \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
c & ::= & F \mid C^0 \mid C^1 \mid C^2 \mid \cdots
\end{array}
$$

Figure 2.2: Expressions $e \in \mathit{EExp}$

each syntactic occurrence of `channel` is assigned a *label* $l$ (taken from some unspecified set *Lab*).

**Source programs** are expressions without any free identifiers, where *expressions* ($e \in \mathit{Exp}$) are given by the syntax in Figure 2.1. We thus require all constructors and base functions to be fully applied; this facilitates the technical development and is no serious restriction as "partial applications" can easily be encoded: instead of writing say `cons` 3 one writes `fn` $x \Rightarrow$ `cons` $< 3, x >$.

We shall allow to write $C^0$ for $C^0 < >$, to write $(e_1, e_2)$ for `pair` $< e_1, e_2 >$, to write `[]` for `nil`, and to write $[e_1, \cdots, e_n]$ for `cons` $< e_1, [e_2, \cdots, e_n] >$. Additionally we shall write $e_1 ; e_2$ for `snd` $< (e_1, e_2) >$; to motivate this notice that since the language is call-by-value, evaluation of the latter expression will give rise to evaluation of $e_1$ followed by evaluation of $e_2$, the value of which will be the final result.

When typing expressions it is convenient to work with *extended expressions* ($e \in \mathit{EExp}$), given by the syntax[4] in Figure 2.2: Compared to Fig. 2.1 the full application of a constructor or base function has been removed, instead constants have become first class objects and a special kind of "silent function application" $e_0 \mathbin{@}^s_n < e_1, \cdots, e_n >$ ($n \geq 1$) has been introduced.

There is a natural injection $\mathcal{T}$ from *Exp* into *EExp* as tabulated in Fig. 2.3, exploiting that application of a constructor and the application of a *sequential base function* takes place "silently" whereas the application of a *non-sequential base function* may have visible (audible!) effect.

[4]In this figure, $e$ ranges over *EExp*.

$$
\begin{array}{lcl}
\mathcal{T}(x) & = & x \\
\mathcal{T}(\texttt{fn}\ x \Rightarrow e) & = & \texttt{fn}\ x \Rightarrow \mathcal{T}(e) \\
\mathcal{T}(e_1\ e_2) & = & \mathcal{T}(e_1)\ \mathcal{T}(e_2) \\
\mathcal{T}(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) & = & \texttt{let}\ x = \mathcal{T}(e_1)\ \texttt{in}\ \mathcal{T}(e_2) \\
\mathcal{T}(\texttt{rec}\ f\ x \Rightarrow e) & = & \texttt{rec}\ f\ x \Rightarrow \mathcal{T}(e) \\
\mathcal{T}(\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2) & = & \texttt{if}\ \mathcal{T}(e)\ \texttt{then}\ \mathcal{T}(e_1)\ \texttt{else}\ \mathcal{T}(e_2) \\
\mathcal{T}(F_s < e >) & = & F_s\ @_1^s\ < \mathcal{T}(e) > \\
\mathcal{T}(F_c < e >) & = & F_c\ \mathcal{T}(e) \\
\mathcal{T}(C^0 < >) & = & C^0 \\
\mathcal{T}(C^1 < e_1 >) & = & C^1\ @_1^s\ < \mathcal{T}(e_1) > \\
\mathcal{T}(C^2 < e_1, e_2 >) & = & C^2\ @_2^s\ < \mathcal{T}(e_1), \mathcal{T}(e_2) >
\end{array}
$$

Figure 2.3: Translating from *Exp* to *EExp*

We shall often identify $e \in \mathit{Exp}$ with $\mathcal{T}(e) \in \mathit{EExp}$; whether $e$ ranges over *Exp* or *EExp* will usually be clear from context.

**Remark** We stated in the Introduction that our development is widely applicable. To this end it is worth pointing out the similarities between the `ref`-types of Standard ML and the delayed communications of Concurrent ML. In particular `ref` $e$ corresponds to `channel` $<$ `()` $>$, $e_1$`:=`$e_2$ corresponds to `sync` $<$ `transmit` $<$ `(`$e_1$`,`$e_2$`)` $>$ $>$, and `!`$e$ corresponds to `sync` $<$ `receive` $< e >$ $>$. Looking slightly ahead the Standard ML type $t$ `ref` will correspond to the Concurrent ML type $t$ `chan` . □

**Example 2.1** The following CML-program *map2* is a version of the well-known *map* function except that a process is spawned for each tail while the spawning process itself works on the head.

```
rec map2 f =>
   fn xs =>
     if null(xs) then []
     else let ch = channel¹ ()
          in spawn (fn d =>
                       (sync (transmit (ch, map2 f (tl xs)))));
             cons (f (hd xs))
                  (sync (receive ch))
```

Let *f* be a function which when applied to an argument of type $\alpha_1$ performs the concurrent actions indicated by $\beta_1$ and at the end returns a value of type $\alpha_2$. Then *map2 f* will be a function which when applied to a list *xs* will perform the following concurrent actions (indicated by $\beta_2$): either it performs no communication (if *xs* is empty) or it will first allocate in region $\{1\}$ a channel which transmits values of type $\alpha_2$ `list`; then it spawns a process which first behaves like $\beta_2$ (to work "recursively" on the tail of the list) and then outputs to region $\{1\}$ a value of type $\alpha_2$ `list`; then it performs $\beta_1$ (when computing $f$ on the head of the list); and finally it receives from region $\{1\}$ a value of type $\alpha_2$ `list`. □

In Section 2.5 we shall see how our inference system enables us to express the information sketched above in a compact way by means of behaviours. This supports a two-stage approach to program analysis: instead of writing a number of analyses for CML programs one writes these analyses for behaviours (presumably a much easier task) and then relies on *one* analysis mapping CML programs into behaviours.

**Example 2.2** Consider the program

```
fn f => let id = fn y =>
                  (if true
                   then f
                   else fn x =>
                         (sync (transmit (channel¹ (), y));
                         x));
                  y
        in id id
```

that takes a function `f` as argument, defines an identity function `id`, and then applies `id` to itself. The identity function contains a conditional whose sole purpose is to force `f` and a locally defined function to have the same type. The locally defined function is yet another identity function except that it attempts to send the argument to `id` over a newly created channel. (To be able to execute one would need to spawn a process that could read over the same channel.)

This program is of interest because it will be rejected by a system using subeffecting only, whereas it will be accepted in the systems of [28] and [30].

In Sect. 2.5 we shall see that we will be able to type this program in our system as well! □

## 2.1 Annotated Types

To prepare for the type inference system we must clarify the syntax of types, behaviours, regions, substitutions, type schemes, and constraints. The syntax of *types* ($t \in \mathit{Typ}$) is given by:

$$\begin{array}{lll} t & ::= & \alpha \mid \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid t_1 \ \rightarrow\ t_2 \mid t_1 \ \rightarrow^{\beta}\ t_2 \\ & \mid & t_1 \times t_2 \mid t\ \texttt{list} \mid t\ \texttt{chan}\ \rho \mid t\ \texttt{event}\ \beta \end{array}$$

that is in addition to type variables (denoted $\alpha$) we have base types including the unit type, booleans and integers; composite types include the function type, the product type and the list type; finally we have the type $t\ \texttt{chan}\ \rho$ for a typed channel allowing values of type $t$ to be transmitted, and the type $t\ \texttt{event}\ \beta$ for a delayed communication that will eventually result in a value of type $t$.

Except for the presence of a $\beta$-component in $t_1 \ \rightarrow^{\beta}\ t_2$ (omitted in a "silent" function type $t_1 \ \rightarrow\ t_2$) and $t\ \texttt{event}\ \beta$, and the presence of a $\rho$-component in $t\ \texttt{chan}\ \rho$, this is much the same type structure that is actually used in Concurrent ML [22]. The role of the *region variable* $\rho$ is to express the origin of the channel, that is the label $l$ of the $\texttt{channel}^l$ call which created it; accordingly the syntax of *regions* ($r \in \mathit{Reg}$) is given by

$$r ::= \rho \mid \{l\}$$

The role of the *behaviour variable* $\beta$ is to express the dynamic effect that takes place when the function is applied or the delayed communication synchronised; motivated by [17] the syntax of *behaviours* ($b \in \mathit{Beh}$) is given by:

$$\begin{array}{lll} b & ::= & \beta \mid \varepsilon \mid b_1; b_2 \mid b_1 + b_2 \\ & \mid & \mathit{SPAWN}\, b \mid t\ \mathrm{CHAN}\ \rho \mid \rho\,!\,t \mid \rho\,?\,t \end{array}$$

that is in addition to behaviour variables we have the empty behaviour $\varepsilon$ (no "visible" actions take place); a sequential composition $b_1; b_2$ (first $b_1$ is performed and then $b_2$); a non-deterministic choice $b_1 + b_2$ (either $b_1$ or $b_2$ are performed); $SPAWN\, b$ (a process with behaviour $b$ is created); $t$ CHAN $\rho$ (a channel able to transmit values of type $t$ is created in region $\rho$); $\rho\,!\,t$ (a value of type $t$ is sent over a channel in region $\rho$); $\rho\,?\,t$ (a value of type $t$ is received over a channel in region $\rho$).

So compared with the effects in e.g. [28] we have (by means of the ; operator) incorporated causality information; on the other hand we have not allowed to mask out behaviours which operate on "inaccessible" regions (cf. Chap. 1). In contrast to [17] there is no explicit recursion; in Section 2.5 we shall see that constraints may implicitly give rise to "recursive" behaviours.

A *substitution* is a mapping from type variables into types and behaviour variables into behaviour *variables* and region variables into region *variables* such that the domain is finite. Here the domain of a substitution $S$ is $Dom(S) = \{\gamma \mid S\,\gamma \neq \gamma\}$ and the range is $Ran(S) = \bigcup\, \{FV(S\,\gamma) \mid \gamma \in Dom(S)\}$, where we use the letter $\gamma$ to range over $\alpha$'s and $\beta$'s and $\rho$'s as appropriate (and similarly we use $g$ to range over $t$'s and $b$'s and $r$'s as appropriate). The identity substitution is denoted Id. The result of composing $S_1$ and $S_2$, i.e. the mapping which takes each $\gamma$ into $S_2(S_1(\gamma))$, is denoted $S_2\, S_1$.

A *constraint set* $C$ is a finite set of type inclusions ($t_1 \subseteq t_2$) and behaviour inclusions ($b_1 \subseteq b_2$) and region inclusions ($r_1 \subseteq r_2$); the set of type inclusions in $C$ will be written $C^t$ and the set of behaviour inclusions in $C$ will be written $C^b$ and the set of region inclusions in $C$ will be written $C^r$.

**Remark** As the result of applying a substitution $S$ to a type must be a (well-defined) type, we had to impose the restriction that $S\,\beta$ must be of the form $\beta'$ (and that $S\,\rho$ must be of the form $\rho'$). Alternatively one could allow types to contain more complex behaviours, permitting say $\texttt{int} \rightarrow^{\rho\,!\,\texttt{int}} \texttt{int}$; the definition chosen amounts to demanding that types should be (what [14] calls) *simple*. When designing a reconstruction algorithm it is apparently a key feature to require all types in question to be simple, as in [27] and [32], but in [27] the inference system employs non-simple types and in [32] a "direct" as well as an "indirect" inference system (the latter geared towards an algorithm employing constraints) is given. We have chosen (also to facilitate the correctness proof of the algorithm) a more uniform approach, perhaps similar in spirit to [31] where arrows are annotated with pairs of the form $\epsilon.\phi$ with $\epsilon$ an effect variable and with $\phi$ a set of region or effect variables: one can think of this as an arrow annotated with $\epsilon$ *together with* the constraint $\phi \subseteq \epsilon$. Similarly we in our framework can "encode" the above "type" $\texttt{int} \rightarrow^{\rho!\texttt{int}} \texttt{int}$ as $\texttt{int} \rightarrow^{\beta} \texttt{int}$ together with the constraint $\rho!\texttt{int} \subseteq \beta$. □

A *type scheme* ($ts \in TSch$) is given by

$$ts \quad ::= \quad \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$$

where $\vec{\alpha}\vec{\beta}\vec{\rho}$ is the list of quantified type and behaviour and region variables, $C$ is a constraint set, and $t$ is the type. We regard type schemes as equivalent up to renaming of bound variables. There is a natural injection[5] from types into type schemes which takes the type $t$ into the type scheme $\forall(() : \emptyset).\ t$. We use the letter $\sigma$ to range over types $t$ and type schemes $ts$ as appropriate.

An *environment* $A$ is a list $[c_1 : \sigma'_1, \cdots, c_m : \sigma'_m, x_1 : \sigma_1, \cdots, x_n : \sigma_n]$ of typing assumptions for constants and identifiers; we let $A(x)$ denote the rightmost entry for $x$ in $A$, similarly for $A(c)$. We shall only deal with *standard* environments, where an environment is standard if on constants it behaves as in Figure 2.4 which we shall motivate briefly:

First notice that all function types are silent except those occurring in non-sequential base functions, cf. the translation in Fig. 2.3. For the sequential constants the constraint set is empty and the type is as in Standard ML. Turning to the non-sequential constants, the type of `sync` interacts closely with the types of `transmit` and `receive`: if `ch` is a channel of type $t$ `chan` $\rho$, the expression $\texttt{receive}\,@_1^s < \texttt{ch} >$ is going to have type $t$ `event` $\beta$ with $\rho?t \subseteq \beta$, and the expression $\texttt{sync}\,(\texttt{receive}\,@_1^s < \texttt{ch} >)$ is going to have type $t$; similarly for `transmit`. The type of $\texttt{channel}^l$ records the type of the created channel as well as its origin $l$ in the annotation of the function type; finally the type of `spawn` records the behaviour of the spawned process. (As discussed previously one might add `wrap` to the language: this constant transforms delayed communications of type $t$ `event` $\beta$ into delayed communications of type $t'$ `event` $\beta'$.)

We will incorporate the effects of [28, 17] into the approach of [26, 8] by defining a type inference system with judgements of the form

[5] We shall distinguish rather sharply between these two entities, but Observation 2.15 suggests that they may be identified.

| $c$ | $A(c)$ |
| --- | --- |
| `()` | $\texttt{unit}$ |
| $\texttt{true}, \texttt{false}$ | $\texttt{bool}$ |
| $\cdots \Leftrightarrow 1, 0, 1, 2 \cdots$ | $\texttt{int}$ |
| `+`, `-`, `*`, `/` | $\texttt{int} \times \texttt{int} \ \rightarrow \ \texttt{int}$ |
| `=` | $\texttt{int} \times \texttt{int} \ \rightarrow \ \texttt{bool}$ |
| `pair` | $\forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \ \rightarrow \ \alpha_2 \ \rightarrow \ \alpha_1 \times \alpha_2$ |
| `fst` | $\forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \times \alpha_2 \ \rightarrow \ \alpha_1$ |
| `snd` | $\forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \times \alpha_2 \ \rightarrow \ \alpha_2$ |
| `nil` | $\forall(\alpha : \emptyset).\ \alpha\ \texttt{list}$ |
| `cons` | $\forall(\alpha : \emptyset).\ \alpha \ \rightarrow \ \alpha\ \texttt{list} \ \rightarrow \ \alpha\ \texttt{list}$ |
| `hd` | $\forall(\alpha : \emptyset).\ \alpha\ \texttt{list} \ \rightarrow \ \alpha$ |
| `tl` | $\forall(\alpha : \emptyset).\ \alpha\ \texttt{list} \ \rightarrow \ \alpha\ \texttt{list}$ |
| `null` | $\forall(\alpha : \emptyset).\ \alpha\ \texttt{list} \ \rightarrow \ \texttt{bool}$ |
| `transmit` | $\forall(\alpha\beta\rho : \{\rho\,!\,\alpha \subseteq \beta\}).\ (\alpha\ \texttt{chan}\ \rho) \times \alpha \ \rightarrow \ (\alpha\ \texttt{event}\ \beta)$ |
| `receive` | $\forall(\alpha\beta\rho : \{\rho\,?\,\alpha \subseteq \beta\}).\ (\alpha\ \texttt{chan}\ \rho) \ \rightarrow \ (\alpha\ \texttt{event}\ \beta)$ |
| `sync` | $\forall(\alpha\beta : \emptyset).\ (\alpha\ \texttt{event}\ \beta) \ \rightarrow^{\beta} \ \alpha$ |
| $\texttt{channel}^{l}$ | $\forall(\alpha\beta\rho : \{\alpha\ \mathrm{CHAN}\ \rho \subseteq \beta,\ \{l\} \subseteq \rho\}).\ \texttt{unit} \ \rightarrow^{\beta} \ (\alpha\ \texttt{chan}\ \rho)$ |
| `spawn` | $\forall(\alpha\beta\beta_0 : \{\mathit{SPAWN}\ \beta_0 \subseteq \beta\}).\ (\texttt{unit} \ \rightarrow^{\beta_0} \ \alpha) \ \rightarrow^{\beta} \ \texttt{unit}$ |

Figure 2.4: The standard types of constants

$$C, A \vdash e : \sigma \,\&\, b$$

where $C$ is a constraint set, $A$ is an environment, $e$ is an expression in *EExp*, $\sigma$ is a type or a type scheme, and $b$ is a behaviour. This means that $e$ has type or type scheme $\sigma$, and that its execution will result in a behaviour described by $b$, assuming that free identifiers and constants have types as specified by $A$ and that all variables are related as described by $C$.

The overall structure of the type inference system of Figure 2.5 is very close to those of [26, 8] with a few components from [28, 17] thrown in; the novel ideas of our approach only show up as carefully constructed side conditions for some of the rules. Concentrating on the "overall picture" we thus have rather straightforward axioms for constants and identifiers: as the language is call-by-value no actions take place when an identifier is retrieved from the environment. The rule for abstraction is largely as usual in effect systems: the latent behaviour of the body of a function abstraction is placed on the arrow of the function type; in our framework this behaviour must be a variable and this can be achieved via subeffecting (Sect. 2.2 and Fig. 2.7).

The rule(s) for application is as one may expect for a call-by-value language: first the function is evaluated, then its argument is evaluated, and finally the function is applied enabling the latent behaviour on the function arrow; in case of a silent function application the function type must be silent (this will hold for expressions belonging to *Exp*, cf. Fig. 2.3 and Fig. 2.4). The rule for `let` is straightforward given that both the `let`-bound expression and the body needs to be evaluated. The rule for recursion makes use of function abstraction to concisely represent the "fixed point requirement" of typing recursive functions; note that we do not admit polymorphic recursion. The rule for conditional is unable to keep track of which branch is chosen, therefore an upper approximation of the branches is taken. We then have separate rules for subtyping, instantiation and generalisation and we shall explain their side conditions in subsequent sections.

## 2.2 Subtyping

Rule (sub) generalises the subeffecting rule of [28] by incorporating subtyping and extends the subtyping rule of [26] to deal with behaviours. To do this we associate three kinds of judgements with a constraint set: the relations

$$\text{(con)} \quad C, A \vdash c : A(c) \,\&\, \varepsilon$$

$$\text{(id)} \quad C, A \vdash x : A(x) \,\&\, \varepsilon$$

$$\text{(abs)} \quad \frac{C, A[x : t_1] \vdash e : t_2 \,\&\, \beta}{C, A \vdash \texttt{fn}\ x \Rightarrow e : (t_1 \ \to^{\beta}\ t_2) \,\&\, \varepsilon}$$

$$\text{(app)} \quad \frac{C, A \vdash e_1 : (t_2 \ \to^{\beta}\ t_1) \,\&\, b_1 \qquad C, A \vdash e_2 : t_2 \,\&\, b_2}{C, A \vdash e_1\ e_2 : t_1 \,\&\, ((b_1; b_2); \beta)}$$

$$\text{(sapp)} \quad \frac{C, A \vdash e_0 : (t_1 \ \to\ \cdots t_n \ \to\ t_0) \,\&\, b_0 \cdots C, A \vdash e_i : t_i \,\&\, b_i \cdots}{C, A \vdash e_0\, @^s_n < e_1, \cdots, e_n > : t_0 \,\&\, (b_0; b_1; \cdots; b_n)}$$

$$\text{(let)} \quad \frac{C, A \vdash e_1 : ts_1 \,\&\, b_1 \qquad C, A[x : ts_1] \vdash e_2 : t_2 \,\&\, b_2}{C, A \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2 \,\&\, (b_1; b_2)}$$

$$\text{(rec)} \quad \frac{C, A[f : t] \vdash \texttt{fn}\ x \Rightarrow e : t \,\&\, b}{C, A \vdash \texttt{rec}\ f\ x \Rightarrow e : t \,\&\, b}$$

$$\text{(if)} \quad \frac{C, A \vdash e_0 : \texttt{bool} \,\&\, b_0 \qquad C, A \vdash e_1 : t \,\&\, b_1 \qquad C, A \vdash e_2 : t \,\&\, b_2}{C, A \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t \,\&\, (b_0; (b_1 + b_2))}$$

$$\text{(sub)} \quad \frac{C, A \vdash e : t \,\&\, b}{C, A \vdash e : t' \,\&\, b'} \qquad \text{if } C \vdash t \subseteq t' \text{ and } C \vdash b \subseteq b'$$

$$\text{(ins)} \quad \frac{C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}{C, A \vdash e : S_0\, t_0 \,\&\, b} \qquad \text{if } \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \text{ is solvable from } C \text{ by } S_0$$

$$\text{(gen)} \quad \frac{C \cup C_0, A \vdash e : t_0 \,\&\, b}{C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b} \qquad \text{if } \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \text{ is both well-formed, solvable from } C \text{, and } \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$$

Figure 2.5: The type inference system

$C \vdash t_1 \subseteq t_2$ and $C \vdash b_1 \subseteq b_2$ and $C \vdash r_1 \subseteq r_2$ are defined by the rules and axioms of Figure 2.6 and Figure 2.7 and Figure 2.8 which are mutually recursive. In all cases we write $\equiv$ for the equivalence induced by the orderings. We shall also write $C \vdash C'$ to mean that $C \vdash g_1 \subseteq g_2$ for all $(g_1 \subseteq g_2)$ in $C'$.

The relation $C \vdash t_1 \subseteq t_2$ expresses the usual notion of subtyping: given the assumptions in $C$, $t_1$ is a more precise approximation than $t_2$. It is induced by the subeffecting relation so unlike e.g. [26] we do not have any ordering on base types, such as $\texttt{int} \subseteq \texttt{real}$; in particular it is contravariant in the argument position of a (silent as well as non-silent) function type. In the case of `chan` note that the type $t$ of $t\ \texttt{chan}\ \rho$ essentially occurs both covariantly (when used in `receive`) and contravariantly (when used in `transmit`); hence we must require that $t \equiv t'$ (and also $\rho \subseteq \rho'$ but not necessarily $\rho' \subseteq \rho$) in order for $t\ \texttt{chan}\ \rho \subseteq t'\ \texttt{chan}\ \rho'$ to hold.

The relation $C \vdash b_1 \subseteq b_2$ states that given the assumptions in $C$, $b_1$ is a more precise approximation than $b_2$ in the sense that any action performed by $b_1$ can also be performed by $b_2$.[6] Its definition[7] expresses that sequential composition "$;$" is associative (seq-ass) with $\varepsilon$ as neutral element (seq-neut); that "$\subseteq$" is a congruence wrt. the various behaviour constructors (cong); and that $+$ is least upper bound wrt. $\subseteq$ (ub,lub). Observe that we have no rules for relating say $\rho!t$ to $\rho!t'$ even if $t \equiv t'$; this is due to technical reasons (in particular the desire that Lemma 2.29 should hold).

In contrast to what is standard in the literature we have explicit rules (bw) for running the structural subtyping rules backwards; enabling us to "decompose" a type constraint into type and behaviour and region constraints. On the other hand it would not make sense to run the behaviour inference system backwards, as $b_1; b_2 \subseteq b_1'; b_2'$ does not entail $b_1 \subseteq b_1'$ and $b_2 \subseteq b_2'$ (consider e.g. $b_1 = b_2' = \varepsilon$ and $b_1' = b_2 = \rho!\texttt{int}$).

[6] A similar claim is formalised in [18] where a syntactically defined ordering on behaviours is shown to be a decidable subset of the undecidable simulation ordering, defined using an operational semantics for behaviours.

[7] One might also add the rule $C \vdash (b_1 + b_2); b_3 \equiv (b_1; b_3) + (b_2; b_3)$.

(axiom) $C \vdash t_1 \subseteq t_2$ if $(t_1 \subseteq t_2) \in C$

(refl) $C \vdash t \subseteq t$

(trans) $$\frac{C \vdash t_1 \subseteq t_2 \quad C \vdash t_2 \subseteq t_3}{C \vdash t_1 \subseteq t_3}$$

($\to$) $$\frac{C \vdash t_1' \subseteq t_1 \quad C \vdash t_2 \subseteq t_2'}{C \vdash (t_1 \to t_2) \subseteq (t_1' \to t_2')}$$

$$\frac{C \vdash t_1' \subseteq t_1 \quad C \vdash t_2 \subseteq t_2' \quad C \vdash \beta \subseteq \beta'}{C \vdash (t_1 \to^{\beta} t_2) \subseteq (t_1' \to^{\beta'} t_2')}$$

($\times$) $$\frac{C \vdash t_1 \subseteq t_1' \quad C \vdash t_2 \subseteq t_2'}{C \vdash (t_1 \times t_2) \subseteq (t_1' \times t_2')}$$

(`list`) $$\frac{C \vdash t \subseteq t'}{C \vdash (t\ \texttt{list}) \subseteq (t'\ \texttt{list})}$$

(`chan`) $$\frac{C \vdash t \equiv t' \quad C \vdash \rho \subseteq \rho'}{C \vdash (t\ \texttt{chan}\ \rho) \subseteq (t'\ \texttt{chan}\ \rho')}$$

(`event`) $$\frac{C \vdash t \subseteq t' \quad C \vdash \beta \subseteq \beta'}{C \vdash (t\ \texttt{event}\ \beta) \subseteq (t'\ \texttt{event}\ \beta')}$$

(bw) $$\frac{C \vdash (t_1 \to^{\beta} t_2) \subseteq (t_1' \to^{\beta'} t_2')}{C \vdash t_1' \subseteq t_1} \qquad \frac{C \vdash (t_1 \to t_2) \subseteq (t_1' \to t_2')}{C \vdash t_1' \subseteq t_1}$$

$$\frac{C \vdash (t_1 \to^{\beta} t_2) \subseteq (t_1' \to^{\beta'} t_2')}{C \vdash t_2 \subseteq t_2'} \qquad \frac{C \vdash (t_1 \to t_2) \subseteq (t_1' \to t_2')}{C \vdash t_2 \subseteq t_2'}$$

$$\frac{C \vdash (t_1 \times t_2) \subseteq (t_1' \times t_2')}{C \vdash t_1 \subseteq t_1'} \qquad \frac{C \vdash (t_1 \times t_2) \subseteq (t_1' \times t_2')}{C \vdash t_2 \subseteq t_2'}$$

$$\frac{C \vdash (t\ \texttt{list}) \subseteq (t'\ \texttt{list})}{C \vdash t \subseteq t'}$$

$$\frac{C \vdash (t\ \texttt{chan}\ \rho) \subseteq (t'\ \texttt{chan}\ \rho')}{C \vdash t \subseteq t'} \qquad \frac{C \vdash (t\ \texttt{chan}\ \rho) \subseteq (t'\ \texttt{chan}\ \rho')}{C \vdash t' \subseteq t}$$

$$\frac{C \vdash (t\ \texttt{event}\ \beta) \subseteq (t'\ \texttt{event}\ \beta')}{C \vdash t \subseteq t'}$$

Figure 2.6: Subtyping

(axiom) $C \vdash b_1 \subseteq b_2$ if $(b_1 \subseteq b_2) \in C$

(refl) $C \vdash b \subseteq b$

(trans) $$\frac{C \vdash b_1 \subseteq b_2 \quad C \vdash b_2 \subseteq b_3}{C \vdash b_1 \subseteq b_3}$$

(cong) $$\frac{C \vdash b_1 \subseteq b_1' \quad C \vdash b_2 \subseteq b_2'}{C \vdash b_1; b_2 \subseteq b_1'; b_2'}$$

$$\frac{C \vdash b_1 \subseteq b_1' \quad C \vdash b_2 \subseteq b_2'}{C \vdash b_1 + b_2 \subseteq b_1' + b_2'}$$

$$\frac{C \vdash b \subseteq b'}{C \vdash \mathit{SPAWN}\, b \subseteq \mathit{SPAWN}\, b'}$$

(seq-ass) $C \vdash b_1; (b_2; b_3) \equiv (b_1; b_2); b_3$

(seq-neut) $C \vdash \varepsilon; b \equiv b$ $C \vdash b; \varepsilon \equiv b$

(ub) $C \vdash b_1 \subseteq b_1 + b_2$ $C \vdash b_2 \subseteq b_1 + b_2$

(lub) $$\frac{C \vdash b_1 \subseteq b \quad C \vdash b_2 \subseteq b}{C \vdash b_1 + b_2 \subseteq b}$$

(bw) $$\frac{C \vdash (t_1 \rightarrow^{\beta} t_2) \subseteq (t_1' \rightarrow^{\beta'} t_2')}{C \vdash \beta \subseteq \beta'}$$

$$\frac{C \vdash (t\ \texttt{event}\ \beta) \subseteq (t'\ \texttt{event}\ \beta')}{C \vdash \beta \subseteq \beta'}$$

Figure 2.7: Subeffecting

| | | |
|---|---|---|
| (axiom) | $C \vdash r_1 \subseteq r_2$ | if $(r_1 \subseteq r_2) \in C$ |
| (refl) | $C \vdash r \subseteq r$ | |
| (trans) | $\dfrac{C \vdash r_1 \subseteq r_2 \quad C \vdash r_2 \subseteq r_3}{C \vdash r_1 \subseteq r_3}$ | |
| (bw) | $\dfrac{C \vdash (t\ \texttt{chan}\ \rho) \subseteq (t'\ \texttt{chan}\ \rho')}{C \vdash \rho \subseteq \rho'}$ | |

Figure 2.8: Subregions

## 2.3 Instantiation

Rule (ins) is much as in [26] and merely says that to take an instance of a type scheme we must ensure that the constraints are satisfied; this is expressed using the notion of *solvability*:

**Definition 2.3** The type scheme $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is *solvable* from $C$ by the substitution $S_0$ if $Dom(S_0) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and if $C \vdash S_0\, C_0$.

A type scheme $ts$ is solvable from $C$ if there exists a substitution $S$ such that $ts$ is solvable from $C$ by $S$.

As $\forall(() : \emptyset).\ t$ is trivially solvable from $C$, we stipulate that a type $t$ is solvable from $C$.

An environment $A$ is solvable from $C$ if it for all $c$ in $Dom(A)$ holds that $A(c)$ is solvable from $C$, and it for all $x$ in $Dom(A)$ holds that $A(x)$ is solvable from $C$. □

**Observation 2.4** As expected we have the following property: if $ts$ and $ts'$ are equivalent up to renaming of bound variables, then everything that can be derived from $C, A \vdash e : ts \,\&\, b$, using (ins), can also be derived from $C, A \vdash e : ts' \,\&\, b$.

**Observation 2.5** Suppose that

$$C, A \vdash e_1 : t_1 \,\&\, b_1 \text{ and } C, A \vdash e_2 : t_2 \,\&\, b_2;$$

since $C, A \vdash \texttt{pair} : t_1 \rightarrow t_2 \rightarrow t_1 \times t_2 \,\&\, \varepsilon$ we clearly have

$$C, A \vdash \texttt{(}e_1\texttt{,}e_2\texttt{)} : t_1 \times t_2 \,\&\, b_1; b_2.$$

By similar reasoning we may arrive at other "derived rules", e.g.

$$\frac{C, A \vdash e_1 : t_1 \,\&\, b_1 \quad C, A \vdash e_2 : t_2 \,\&\, b_2}{C, A \vdash e_1\texttt{;}e_2 : t_2 \,\&\, b_1; b_2}$$

## 2.4 Generalisation

Except for the well-formedness requirement (explained later), rule (gen) seems close to the corresponding rule in [26]: clearly we cannot generalise over variables free in the global type assumptions or global constraint sets, and as in effect systems (e.g. [28]) we cannot generalise over variables visible in the effect. Furthermore, as in [26] solvability is imposed to ensure that we do not create type schemes that have no instances; this condition ensures that the expressions `let x =` $e_1$ `in` $e_2$ and `let x =` $e_1$ `in (x;`$e_2$`)` are going to be equivalent in the type system.

**Example 2.6** Without an additional notion of well-formedness this does not give a semantically sound rule (gen); as an example consider the expression $e$ given by

```
let ch = channel¹ ()
in  ···
    (sync(transmit(ch,7)))
    (sync(transmit(ch,true)))
```

and note that it is semantically unsound (at least if "$\cdots$" spawned some process receiving twice over `ch` and adding the results). Writing $C = \{\{1\} \subseteq \rho,\ \alpha \text{ CHAN } \rho \subseteq \beta,\ \texttt{int} \text{ CHAN } \rho \subseteq \beta,\ \texttt{bool} \text{ CHAN } \rho \subseteq \beta\}$ and $C' = \{\alpha' \text{ CHAN } \rho \subseteq \beta\}$ gives (with $A$ standard)

$$C \cup C', A \vdash \texttt{channel}^1 : \texttt{unit} \rightarrow^{\beta} \alpha' \texttt{ chan } \rho \,\&\, \varepsilon$$

and therefore

$$C \cup C', A \vdash \texttt{channel}^1 \texttt{ ()} : \alpha' \texttt{ chan } \rho \,\&\, \beta$$

and, without taking well-formedness into account, rule (gen) would give

$$C, A \vdash \texttt{channel}^1\ \texttt{()}\ :\ (\forall(\alpha' : C').\ \alpha'\ \texttt{chan}\ \rho)\ \&\ \beta$$

because $\alpha' \notin FV(C, A, \beta)$ and $\forall(\alpha' : C').\ \alpha'\ \texttt{chan}\ \rho$ is solvable from $C$ by either of the substitutions $[\alpha' \mapsto \alpha]$, $[\alpha' \mapsto \texttt{int}]$ and $[\alpha' \mapsto \texttt{bool}]$. This then would give

$$C, A[\texttt{ch} : \forall(\alpha' : C').\ \alpha'\ \texttt{chan}\ \rho] \vdash \texttt{ch}\ :\ \texttt{int chan}\ \rho\ \&\ \varepsilon$$
$$C, A[\texttt{ch} : \forall(\alpha' : C').\ \alpha'\ \texttt{chan}\ \rho] \vdash \texttt{ch}\ :\ \texttt{bool chan}\ \rho\ \&\ \varepsilon$$

so that

$$C, A \vdash e\ :\ t\ \&\ b$$

for suitable $t$ and $b$. As the constraint set $C$ does not in any way seem "unreasonable" or "inconsistent", this shows that some notion of well-formedness (for type schemes) is essential for semantic soundness; actually the example suggests that if there is a constraint $(\alpha'\ \text{CHAN}\ \rho \subseteq \beta)$ then one should not generalise over $\alpha'$ if it is impossible to generalise over $\beta$. □

### 2.4.1 The Arrow Relation

In order to formalise the notion of well-formedness, we next associate another kind of judgement and two kinds of closure with a constraint set. In order to do so, we employ the notion of *backwards closure*:

**Definition 2.7** Let $C$ be a constraint set. Then the backwards closure of $C$, written $\overline{C}$, is defined as

$$\overline{C} = \{(g_1 \subseteq g_2) \mid C \vdash_{dc} g_1 \subseteq g_2\}$$

where $\vdash_{dc}$ denotes a derivation which uses only the rules (axiom) in Figs. 2.6 and 2.7 and 2.8, the rule (trans) in Fig. 2.6 (but *not* in Fig. 2.7 or 2.8), and the rules (bw) in Figs. 2.6 and 2.7 and 2.8. □

So $\overline{C}$ is the least set containing $C$ which is closed under decomposition of type constraints and under transitive closure of the type constraints; it thus holds that $\overline{C} = \overline{C^t} \cup C^b \cup C^r$. Notice that if $(g_1 \subseteq g_2) \in \overline{C}$ then $g_1$ as well as $g_2$ will be a syntactic subpart of $C$, implying that if $C$ is finite then $\overline{C}$ is finite and that $FV(\overline{C}) = FV(C)$.

Motivated by the concluding remark of Example 2.6 we now establish a relation between the right hand side variable and the left hand side variables in a constraint $b \subseteq \beta$:

**Definition 2.8** The judgement $C \vdash \gamma \leftarrow \beta$ holds iff there exists $(b \subseteq \beta)$ in $\overline{C}$ such that $\gamma \in FV(b)$. □

**Remark** Alternatively one could define that $C \vdash \gamma \leftarrow \beta$ holds iff there exists $(b \subseteq \beta)$ in $\overline{C}$ such that $\gamma \in topchan(b)$, where $topchan(b)$ are those variables which occur in a part of $b$ not inside some $\rho!t$ or $\rho?t$. This would formalise the intuition that it is *channel allocation* (not read and write) which is "dangerous", cf. the discussion in the Introduction. We conjecture that the future development will carry through using this revised definition with some obvious modifications; but as it is not clear whether it will really add to the power of the type system and as it will add a further level of complexity to the exhibition, we shall refrain from such an attempt. □

**Definition 2.9** For a set $X$ of variables the downwards closure $X^{C\downarrow}$ and the upwards closure $X^{C\uparrow}$ is given by:

$$\begin{array}{rcl} X^{C\downarrow} & = & \{\gamma_1 \mid \exists \gamma_2 \in X : C \vdash \gamma_1 \leftarrow^* \gamma_2\} \\ X^{C\uparrow} & = & \{\gamma_1 \mid \exists \gamma_2 \in X : C \vdash \gamma_2 \leftarrow^* \gamma_1\} \end{array}$$ □

(As usual, $\leftarrow^*$ denotes the reflexive and transitive closure of $\leftarrow$.) It is instructive to think of $C \vdash \gamma_1 \leftarrow \gamma_2$ as defining a directed graph structure upon $FV(C)$; then $X^{C\downarrow}$ is the reachability closure of $X$ and $X^{C\uparrow}$ is the reachability closure in the graph where all edges are reversed.

### 2.4.2 Well-formedness

We can now define the notion of well-formedness for constraints and for type schemes; for the latter we make use of the arrow relations defined above.

**Definition 2.10** *Well-formed constraint sets*

A constraint set $C$ is *well-formed* if all behaviour constraints in $C$ are of the form $b \subseteq \beta$ and if all region constraints in $C$ are of the form $r \subseteq \rho$. $\square$

Requiring the right hand side of a behaviour constraint to be a variable is crucial for our development and is motivated by a desire to represent the constraints in a form such that it is easy to "read a solution": consider for instance the constraint set $\{b_1 \subseteq \beta, b_2 \subseteq \beta\}$ which is equivalent to the constraint set $\{b_1 + b_2 \subseteq \beta\}$ and this suggests that one should really "interpret" $\beta$ as $b_1 + b_2$. On the other hand, we do not know how to handle say the constraint set $\{\beta \subseteq b_1, \beta \subseteq b_2\}$ as we have no explicit "greatest lower bound operator". Similarly, given the constraint set $\{\{1\} \subseteq \rho, \{2\} \subseteq \rho\}$ we should really "interpret" $\rho$ as the *set* $\{1, 2\}$.

**Fact 2.11** Let $C$ be well-formed. Then $\overline{C}$ is well-formed; and for all substitutions $S$ also $S\,C$ is well-formed. $\square$

We now turn to well-formedness of type schemes where we ensure that the embedded constraints are themselves well-formed. Additionally we shall wish to ensure that the set of variables over which we generalise is sensibly related to the constraints (unlike the situation in Example 2.6). The key idea is that if $C \vdash \gamma \leftarrow \beta$ then we do not generalise over $\gamma$ unless we also generalise over $\beta$. These considerations lead to:

**Definition 2.12** *Well-formed type schemes*

A type scheme $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ is *well-formed* if the following conditions hold:

1. $C$ is well-formed;
2. all $(g_1 \subseteq g_2)$ in $C$ contain at least one variable among $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}$;
3. $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ is upwards closed, i.e. $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{C\uparrow}$;
4. $FV(C^t) \cap \{\vec{\beta}\} = \emptyset$.

A type $t$ is trivially well-formed. $\square$

Notice that if $C = \emptyset$ then $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ is well-formed, motivating why all types are well-formed. Requirement 4 is needed in order for the following essential closedness property:

**Fact 2.13** *Well-formedness and Substitutions*

If $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ is well-formed then also $S\,(\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t)$ is well-formed (for all substitutions $S$).

**Proof** We can, without loss of generality, assume that $(Dom(S) \cup Ran(S)) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$. Then $S\,(\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t) = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : S\,C).\ S\,t$. By Fact 2.11 we see that Requirement 1 will still hold; so as Requirements 2 and 4 are clearly fulfilled it suffices to show that $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{S\,C\uparrow}$, i.e. that if $\gamma \in \vec{\alpha}\vec{\beta}\vec{\rho}$ and $S\,C \vdash \gamma \leftarrow \beta$ then $\beta \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$.

The situation thus is that there exists $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in \overline{S\,C}$; that is either $(b \subseteq \beta) \in \overline{S\,C^t}$ or $(b \subseteq \beta) \in S\,C^b$. But the former is impossible: to see this, observe that all behaviour constraints in $\overline{S\,C^t}$ are of the form $\beta_1 \subseteq \beta_2$, where (by Requirement 4) $\{\beta_1, \beta_2\} \cap \{\vec{\beta}\,\} = \emptyset$.

So it must be the case that $(b \subseteq \beta) \in S\,C^b$; that is there exists $b'$ with $S\,b' = b$ and $\beta'$ with $S\,\beta' = \beta$ such that $(b' \subseteq \beta') \in C^b$. As $Ran(S) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$ we infer that $\gamma \in FV(b')$, implying that $C \vdash \gamma \leftarrow \beta'$. Since $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ is upwards closed we have $\beta' \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$, so as $Dom(S) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$ we have $\beta = S\,\beta' = \beta' \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ as desired. □

**Example 2.14** Continuing Example 2.6 note that $\{\alpha'\}^{C'\uparrow} = \{\alpha', \beta\}$ showing that our current notion of well-formedness prevents the erroneous typing. □

**Observation 2.15** $C, A \vdash e : t\,\&\,b$ holds iff $C, A \vdash e : \forall(() : \emptyset).\ t\,\&\,b$ holds, as (gen) or (ins) can be used to conclude one of them from the other.

## 2.5 Working with the Inference System

In this section we shall explain in some detail how the programs in Example 2.1 and Example 2.2 can be typed using the inference system from Fig. 2.5 (in Chap. 4 we shall present an algorithm which is able to find such typings automatically); and at the end we briefly compare with other approaches.

## Typing the program of Example 2.1

We shall see that by letting $C$ contain the constraints

$$\begin{array}{rcl}
\{1\} & \subseteq & \rho \\
\varepsilon & \subseteq & \beta_e \\
\varepsilon + \beta_c; \beta_F; \beta_1; \beta_r & \subseteq & \beta_2 \\
(\alpha_2\ \texttt{list})\ \text{CHAN}\ \rho & \subseteq & \beta_c \\
\mathit{SPAWN}\ \beta_f & \subseteq & \beta_F \\
\beta_e; \beta_2; \beta_s & \subseteq & \beta_f \\
\rho\,!\,(\alpha_2\ \texttt{list}) & \subseteq & \beta_s \\
\rho\,?\,(\alpha_2\ \texttt{list}) & \subseteq & \beta_r
\end{array}$$

and with $t = (\alpha_1 \ \to^{\beta_1}\ \alpha_2) \ \to^{\beta_e}\ (\alpha_1\ \texttt{list} \ \to^{\beta_2}\ \alpha_2\ \texttt{list})$ it holds that

$$C, A \vdash \mathit{map2} \ :\ t \,\&\, \varepsilon \tag{1}$$

(where $A$ is as in Figure 2.4). The behaviour constraints can be post-processed, using the techniques described in Chap. 6, and as a result we end up with a single behaviour constraint

$$\begin{array}{rl}
& \varepsilon + ((\alpha_2\ \texttt{list})\ \text{CHAN}\ \rho; \mathit{SPAWN}\,(\beta_2; \rho\,!\,(\alpha_2\ \texttt{list})); \beta_1; \rho\,?\,(\alpha_2\ \texttt{list})) \\
\subseteq & \beta_2
\end{array}$$

which shows that we can give $\beta_2$ the following "recursive interpretation" that formalises the explanation in Example 2.1:

$$\begin{array}{rl}
& \varepsilon \\
+ & (\alpha_2\ \texttt{list})\ \text{CHAN}\ \{1\}; \mathit{SPAWN}\,(\beta_2; \{1\}\,!\,(\alpha_2\ \texttt{list})); \beta_1; \{1\}\,?\,(\alpha_2\ \texttt{list})
\end{array}$$

We are left with the task of proving (1). Let $A_1$ be an extension of $A$ where `map2` is bound to $t$ and where `f` is bound to $\alpha_1 \ \to^{\beta_1}\ \alpha_2$; then it will suffice to show

$$C, A_1 \vdash \texttt{fn xs => ...} \ :\ \alpha_1\ \texttt{list} \ \to^{\beta_2}\ \alpha_2\ \texttt{list} \,\&\, \beta_e.$$

Let $A_2$ be an extension of $A_1$ where `xs` is bound to $\alpha_1$ `list`; then it will suffice to show

$$C, A_2 \vdash \texttt{if null(xs) ...} : \alpha_2 \ \texttt{list} \,\&\, \beta_2$$

and as $C, A_2 \vdash \texttt{null(xs)} : \texttt{bool} \,\&\, \varepsilon$ and $C, A_2 \vdash \texttt{[]} : \alpha_2 \ \texttt{list} \,\&\, \varepsilon$ it will suffice to show

$$C, A_2 \vdash \texttt{let ch = ...} : \alpha_2 \ \texttt{list} \,\&\, \beta_c; \beta_F; \beta_1; \beta_r.$$

Let $A_3$ be an extension of $A_2$ where `ch` is bound to $(\alpha_2 \ \texttt{list}) \ \texttt{chan} \ \rho$; as clearly $C, A_2 \vdash \texttt{channel}^1 \ \texttt{()} : (\alpha_2 \ \texttt{list}) \ \texttt{chan} \ \rho \,\&\, \beta_c$ it will suffice to show

$$C, A_3 \vdash \texttt{spawn (...); cons ...} : \alpha_2 \ \texttt{list} \,\&\, \beta_F; \beta_1; \beta_r$$

which (cf. Observation 2.5) can be done by demonstrating

$$C, A_3 \vdash \texttt{spawn (...)} : \texttt{unit} \,\&\, \beta_F \tag{2}$$

$$C, A_3 \vdash \texttt{cons ...} : \alpha_2 \ \texttt{list} \,\&\, \beta_1; \beta_r. \tag{3}$$

To establish (2) it will suffice to show

$$C, A_3 \vdash \texttt{fn d => ...} : \texttt{unit} \ \to^{\beta_f} \ \alpha_2 \ \texttt{list} \,\&\, \varepsilon$$

and with $A_4$ an extension of $A_3$ where `d` is bound to `unit` it will suffice to show

$$C, A_4 \vdash \texttt{sync (transmit ...)} : \alpha_2 \ \texttt{list} \,\&\, \beta_f$$

and to do so it will suffice to show

$$C, A_4 \vdash \texttt{transmit (ch,...)} : (\alpha_2 \ \texttt{list}) \ \texttt{event} \ \beta_s \,\&\, \beta_e; \beta_2$$

which since $C \vdash \rho\,!\,(\alpha_2 \ \texttt{list}) \subseteq \beta_s$ can be done by showing

$$C, A_4 \vdash \texttt{(ch,map2 ...)} : (\alpha_2 \ \texttt{list}) \ \texttt{chan} \ \rho \times \alpha_2 \ \texttt{list} \,\&\, \beta_e; \beta_2$$

and (cf. Observation 2.5) this follows from

$$C, A_4 \vdash \texttt{map2 f (tl xs)} : \alpha_2 \ \texttt{list} \,\&\, \beta_e; \beta_2$$

which is a consequence of the assumptions in $A_4$.

To establish (3) it will suffice to show

$$C, A_3 \vdash \texttt{f (hd xs)} : \alpha_2 \,\&\, \beta_1 \text{ and}$$

$$C, A_3 \vdash \texttt{sync (receive ch)} : \alpha_2\ \texttt{list} \,\&\, \beta_r.$$

The former is an easy consequence of the assumptions in $A_3$; and the latter follows since $C \vdash \rho\,?\,(\alpha_2\ \texttt{list}) \subseteq \beta_r$ and hence

$$C, A_3 \vdash \texttt{receive ch} : (\alpha_2\ \texttt{list})\ \texttt{event}\ \beta_r \,\&\, \varepsilon.$$

## Typing the program of Example 2.2

We shall now explain why this program is accepted by our system. Let

$$C = \{\alpha_y\ \text{CHAN}\ \rho \subseteq \beta_1,\ \rho\,!\,\alpha_y \subseteq \beta_2,\ \beta_1;\beta_2 \subseteq \beta,\ \beta_e \subseteq \beta,\ \{1\} \subseteq \rho\}$$

and let $C' = S_1\, C \cup S_2\, C \cup \{\varepsilon \subseteq \beta_e\}$ with

$$S_1 = [\alpha_y \beta \beta_1 \beta_2 \rho \mapsto \texttt{int}\beta' \beta_1' \beta_2' \rho'] \text{ and}$$

$$S_2 = [\alpha_y \beta \beta_1 \beta_2 \rho \mapsto (\texttt{int} \ \rightarrow^{\beta_e}\ \texttt{int})\beta' \beta_1' \beta_2' \rho'].$$

Let $A$ be as in Fig. 2.4, let $A_f = A[\texttt{f} : \alpha_x \ \rightarrow^{\beta_e}\ \alpha_x]$, let $A_{fy} = A_f[\texttt{y} : \alpha_y]$, and let $A_{fyx} = A_{fy}[\texttt{x} : \alpha_x]$. Finally, let

$$ts = \forall(\alpha_y \beta \beta_1 \beta_2 \rho : C).\ \alpha_y \ \rightarrow^{\beta_e}\ \alpha_y.$$

We shall establish that

$$C', A \vdash \texttt{fn f => ...} : (\alpha_x \ \rightarrow^{\beta_e}\ \alpha_x) \ \rightarrow^{\beta_e}\ (\texttt{int} \ \rightarrow^{\beta_e}\ \texttt{int}) \,\&\, \varepsilon$$

and to do so it will suffice to show

$$C', A_f \vdash \texttt{let id = fn y => ...in id id} : \texttt{int} \ \rightarrow^{\beta_e}\ \texttt{int} \,\&\, \beta_e$$

which can be done by showing

$$C', A_f \vdash \texttt{fn y => ...} : ts \,\&\, \varepsilon \tag{4}$$

$$C', A_f[\texttt{id} : ts] \vdash \texttt{id id} : \texttt{int} \rightarrow^{\beta_e} \texttt{int} \,\&\, \beta_e. \tag{5}$$

To establish (5), we can use $S_2$ and $S_1$ as instance substitutions (as $C' \vdash S_i\, C$ for $i = 1, 2$) to get

$$C', A_f[\texttt{id} : ts] \vdash \texttt{id} : (\texttt{int} \rightarrow^{\beta_e} \texttt{int}) \rightarrow^{\beta_e} (\texttt{int} \rightarrow^{\beta_e} \texttt{int}) \,\&\, \varepsilon$$

$$C', A_f[\texttt{id} : ts] \vdash \texttt{id} : \texttt{int} \rightarrow^{\beta_e} \texttt{int} \,\&\, \varepsilon.$$

It is easy to verify that $ts$ is well-formed, in particular it is upwards closed, and that $\{\alpha_y, \beta, \beta_1, \beta_2, \rho\} \cap FV(C', A_f, \varepsilon) = \emptyset$, in particular observe that

$$\alpha_y \notin FV(A_f(f)) = FV(\alpha_x \rightarrow^{\beta_e} \alpha_x). \tag{6}$$

As it also holds that $ts$ is solvable from $C'$ (by $S_1$ or $S_2$), we can use (gen) to establish (4) if we can show

$$C' \cup C, A_f \vdash \texttt{fn y => ...} : \alpha_y \rightarrow^{\beta_e} \alpha_y \,\&\, \varepsilon$$

which (as $C' \vdash \varepsilon \subseteq \beta_e$) can be done by showing

$$C' \cup C, A_{fy} \vdash \texttt{if ...; y} : \alpha_y \,\&\, \varepsilon$$

which (cf. Observation 2.5) can be done by demonstrating

$$C' \cup C, A_{fy} \vdash \texttt{if true then f else fn x => ...} : \alpha_x \rightarrow^{\beta} \alpha_x \,\&\, \varepsilon$$

$$C' \cup C, A_{fy} \vdash \texttt{y} : \alpha_y \,\&\, \varepsilon.$$

The latter is trivial; and to establish the former it will suffice to show that

$$C' \cup C, A_{fy} \vdash \texttt{f} : \alpha_x \rightarrow^{\beta} \alpha_x \,\&\, \varepsilon \tag{7}$$

$$C' \cup C, A_{fy} \vdash \texttt{fn x => (sync ...; x)} : \alpha_x \rightarrow^{\beta} \alpha_x \,\&\, \varepsilon. \tag{8}$$

(7) can be established by subtyping, since

$$C \vdash \alpha_x \rightarrow^{\beta_e} \alpha_x \subseteq \alpha_x \rightarrow^{\beta} \alpha_x. \qquad (9)$$

To establish (8) it will suffice to show that

$$C' \cup C, A_{fyx} \vdash \texttt{sync (transmit ...); x} : \alpha_x \,\&\, \beta_1; \beta_2$$

which (cf. Observation 2.5) can be done by demonstrating

$$C' \cup C, A_{fyx} \vdash \texttt{sync (transmit ...)} : \alpha_y \,\&\, \beta_1; \beta_2$$

$$C' \cup C, A_{fyx} \vdash \texttt{x} : \alpha_x \,\&\, \varepsilon.$$

The latter is trivial; and in order to establish the former it will be sufficient to show

$$C' \cup C, A_{fyx} \vdash \texttt{transmit (channel}^1 \texttt{ (),y)} : \alpha_y \texttt{ event } \beta_2 \,\&\, \beta_1$$

and this can be done by showing that

$$C' \cup C, A_{fyx} \vdash \texttt{(channel}^1 \texttt{ (),y)} : (\alpha_y \texttt{ chan } \rho) \times \alpha_y \,\&\, \beta_1$$

which (cf. Observation 2.5) is a consequence of

$$C' \cup C, A_{fyx} \vdash \texttt{channel}^1 \texttt{ ()} : \alpha_y \texttt{ chan } \rho \,\&\, \beta_1$$

$$C' \cup C, A_{fyx} \vdash \texttt{y} : \alpha_y \,\&\, \varepsilon.$$

## Other approaches

We have demonstrated that the program from Example 2.2 can be typed in our system, where the subtyping rule was used to establish (9); we shall now examine how other type systems behave on this program.

First consider a system similar to [32] in that *(i)* it employs subeffecting only, and *(ii)* it contains no constraints, so all behaviour information has to be explicitly coded into the types. As $\alpha_y$ (the type of y) is then present in the type of the locally defined function

```
fn x => (sync (transmit (channel¹ (), y)); x)
```

it must also be the case, in order for the two branches in the conditional to match, that $\alpha_y$ is present in the type of `f` (compare with (6)). This means that while the defining expression for `id` still may be assigned the type $\alpha_y \rightarrow^{\varepsilon} \alpha_y$ we are unable to generalise over $\alpha_y$; consequently the application of `id` to itself cannot be typed. (It is interesting to point out that if one changed the applied occurrence of `f` in the program to the expression `fn z => f z` then subeffecting would suffice for generalising over $\alpha_y$ and hence would allow to type the self-application of `id`.)

The system of [28] does not have subtyping but nevertheless the application of `id` to itself is typeable [28, section 11, the case (`id4 id4`)]. This is due to the presence of *regions* and *masking* (cf. the discussion in Chap. 1): with $\rho$ the region in which the new channel is allocated, the expression `sync (transmit (channel (), y))` does not contain $\rho$ in its type $\alpha_y$ and neither is $\rho$ present in the environment, so it is possible to discard the effects $\alpha_y$ CHAN $\rho$ and $\rho\,!\,\alpha_y$. Thus the two branches of the conditional will match.

Also in the approach of [30] one can generalise over $\alpha_y$ and hence type the self-application of `id`. To see this, first note that $\alpha_y$ is classified as an imperative type variable (rather than an applicative type variable which would directly have allowed the generalisation) because $\alpha_y$ is used in the channel construct and thus has a side effect. Despite of this, next note that the defining expression for the `id` function is classified as non-expansive (rather as expansive which would directly have prohibited the generalisation of imperative type variables) because all side effects occurring in the definition of `id` are "protected" by a function abstraction and hence not "dangerous". We refer to [30] for the details.

## 2.6 Basic Properties of the Inference System

We now list a few basic properties of the inference system that we shall use later.

**Fact 2.16** Let $A$ be standard (i.e. on constants it behaves as indicated by Figure 2.4). Then for all constants $c$ the type (scheme) $A(c)$ is closed, well-formed, and satisfies that

- if $c$ is a sequential base function, then (the type part of) $A(c)$ takes the form $t'_1 \rightarrow t'$;

- if $c$ is a constructor $C^n$, then (the type part of) $A(c)$ takes the form $t'_1 \rightarrow \cdots t'_n \rightarrow t'$ with $t'$ not a variable and not a (silent or non-silent) function type. □

So constructors actually construct something (that is, a composite non-functional type).

**Fact 2.17** If $C, A \vdash e : \sigma \,\&\, b$ then

- if $A$ is well-formed then $\sigma$ is well-formed;
- if $A$ is solvable form $C$ then $\sigma$ is solvable from $C$.

**Proof** A straightforward case analysis on the last rule applied. □

**Lemma 2.18** *Substitution Lemma*

For all substitutions $S$:

(a) If $C \vdash C_0$ then $S\,C \vdash S\,C_0$ (and has the same shape).

(b) If $C, A \vdash e : \sigma \,\&\, b$ then $S\,C, S\,A \vdash e : S\,\sigma \,\&\, S\,b$ (and has the same shape).

**Proof** See Appendix A. □

Here the shape of an inference tree is the result of replacing all judgements with the (name of the) axiom or inference rule used to derive them (and dispensing with side conditions).

**Lemma 2.19** *Entailment Lemma*

For all sets $C'$ of constraints satisfying $C' \vdash C$:

(a) If $C \vdash C_0$ then $C' \vdash C_0$.

(b) If $C, A \vdash e : \sigma \,\&\, b$ then $C', A \vdash e : \sigma \,\&\, b$ (and has the same shape).

**Proof** See Appendix A. □

**Fact 2.20** Let $x$,$y$ be distinct: if $C, A_1[x : \sigma_1][y : \sigma_2]A_2 \vdash e : \sigma \& b$ then $C, A_1[y : \sigma_2][x : \sigma_1]A_2 \vdash e : \sigma \& b$ (and has the same shape).

**Fact 2.21** Let $x$ be an identifier not occurring in $e$ and let $t$ be an arbitrary type; if $C, A \vdash e : \sigma \& b$ then $C, A[x : t] \vdash e : \sigma \& b$ (and has the same shape).

**Proof** Let $\alpha$ be a fresh type variable. Then a straightforward induction in the proof tree (using Fact 2.20) tells us that $C, A[x : \alpha] \vdash e : \sigma \& b$ (and has the same shape). Now apply Lemma 2.18 with the substitution $[\alpha \mapsto t]$. □

## 2.7 Proof Normalisation

It turns out that the proof of semantic soundness as well as the proof of completeness of an inference algorithm is complicated by the presence of the non-syntax directed rules (sub), (ins) and (gen) of Figure 2.5. This motivates trying to normalise general inference trees into a more manageable shape:

**Definition 2.22** *Normalisation*

An inference tree for $C, A \vdash e : t \& b$ is *T-normalised* if it is created by:

- (con) or (id); or
- (ins) applied to (con) or (id); or
- (abs), (app), (sapp), (rec), (if) or (sub) applied to T-normalised inference trees; or
- (let) applied to a TS-normalised inference tree and a T-normalised inference tree.

An inference tree for $C, A \vdash e : ts \& b$ is *TS-normalised* if it is created by:

- (gen) applied to a T-normalised inference tree.

We shall write $C, A \vdash_n e : \sigma \& b$ if the inference tree is T-normalised (if $\sigma$ is a type) or TS-normalised (if $\sigma$ is a type scheme). □

Notice that if $jdg = C, A \vdash e : \sigma \,\&\, b$ occurs in a normalised inference tree then we in fact have $jdg = C, A \vdash_n e : \sigma \,\&\, b$, unless $jdg$ is created by (con) or (id) and $\sigma$ is a type scheme.

**Lemma 2.23** Suppose that

$$jdg = C, A \vdash e : S\, t_0 \,\&\, b$$

follows by an application of (ins) to the normalised judgement

$$jdg' = C, A \vdash_n e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$$

where $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \vdash S\, C_0$. Then also $jdg$ has a normalised inference tree:

$$C, A \vdash_n e : S\, t_0 \,\&\, b.$$

**Proof** The TS-normalised judgement $jdg'$ follows by an application of (gen) to the T-normalised judgement

$$C \cup C_0, A \vdash_n e : t_0 \,\&\, b$$

where $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$. From Lemma 2.18 we therefore get

$$C \cup S\, C_0, A \vdash_n e : S\, t_0 \,\&\, b$$

and using Lemma 2.19 we get $C, A \vdash_n e : S\, t_0 \,\&\, b$ as desired. □

**Lemma 2.24** *Normalisation Lemma*

If $A$ is well-formed and solvable from $C$ then an inference tree $C, A \vdash e : \sigma \,\&\, b$ can be transformed into one $C, A \vdash_n e : \sigma \,\&\, b$ that is normalised.

**Proof** See Appendix A. □

## 2.8 Conservative Extension

We next show that our inference system is a conservative extension of the system for (pure functional) ML type inference. For this purpose we restrict ourselves to consider *sequential* expressions only, that is expressions without the non-sequential constructors $C_c^n$ and without the non-sequential base functions $F_c$.

An ML type $u$ (as opposed to a CML type $t$, in the following just denoted type) is either a type variable $\alpha$, a base type like `int`, a function type $u_1 \rightarrow u_2$, a product type $u_1 \times u_2$, or a list type $u_1$ `list`. An ML type scheme $us$ is of the form $\forall \vec{\alpha}.u$. An ML substitution $R$ maps type variables into ML types. Our variant of the ML type inference system is depicted in Figure 2.9, assigning types to sequential expressions in *EExp*. Also here we introduce the notion of *normalised* inferences, denoted $A' \vdash_n^{\mathrm{ML}} e : u$; the definition being a straightforward modification of Def. 2.22, bearing in mind that (sub) is not applicable.

We say that a type is *sequential* if it does not contain subtypes of the form $t$ `chan` $\rho$ or $t$ `event` $\beta$. From a sequential type $t$ we can in a natural way construct an ML type $\epsilon(t)$; it is convenient also to define $\epsilon(t)$ for non-sequential types so we stipulate the total function $\epsilon()$ as follows (where the last clause is somewhat arbitrary): $\epsilon(\alpha) = \alpha$, $\epsilon(\texttt{unit}) = \texttt{unit}$, $\epsilon(\texttt{bool}) = \texttt{bool}$, $\epsilon(\texttt{int}) = \texttt{int}$, $\epsilon(t_1 \rightarrow^\beta t_2) = \epsilon(t_1 \rightarrow t_2) = \epsilon(t_1) \rightarrow \epsilon(t_2)$, $\epsilon(t_1 \times t_2) = \epsilon(t_1) \times \epsilon(t_2)$, $\epsilon(t_1\ \texttt{list}) = \epsilon(t_1)\ \texttt{list}$, and $\epsilon(t\ \texttt{event}\ \beta) = \epsilon(t\ \texttt{chan}\ \rho) = \epsilon(t)$.

We say that a type scheme $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ is *sequential* if $C$ as well as $\vec{\beta}$ and $\vec{\rho}$ is empty and if $t$ is sequential. From a sequential type scheme $ts = \forall(\vec{\alpha} : \emptyset).\ t$ we construct an ML type scheme $\epsilon(ts)$ as follows: $\epsilon(ts) = \forall \vec{\alpha}.\epsilon(t)$. (We shall dispense with defining $\epsilon(ts)$ on non-sequential type schemes for reasons to be discussed in Appendix A.)

Clearly $A(c)$ is sequential for all sequential $c$ if $A$ is as in Fig. 2.4.

Let $\beta$ be a behaviour variable: we say that a sequential type $t$ is *$\beta$-sequential* if no other behaviour variables than $\beta$ occur in $t$; we say that a sequential type scheme $\forall(\vec{\alpha} : \emptyset).\ t$ is $\beta$-sequential if $t$ is $\beta$-sequential; and we let $C_\beta$ denote the constraint set $\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}$.

We are now ready to state that our system conservatively extends ML:

**Theorem 2.25** Let $e$ be a closed sequential expression $\in$ *Exp*. Let $A$ be

$$(\text{con}) \quad A' \vdash^{\mathrm{ML}} c : A'(c)$$

$$(\text{id}) \quad A' \vdash^{\mathrm{ML}} x : A'(x)$$

$$(\text{abs}) \quad \frac{A'[x : u_1] \vdash^{\mathrm{ML}} e : u_2}{A' \vdash^{\mathrm{ML}} \texttt{fn}\; x \Rightarrow e : u_1 \to u_2}$$

$$(\text{app}) \quad \frac{A' \vdash^{\mathrm{ML}} e_1 : u_2 \to u_1, \; A' \vdash^{\mathrm{ML}} e_2 : u_2}{A' \vdash^{\mathrm{ML}} e_1\; e_2 : u_1}$$

$$(\text{sapp}) \quad \frac{A' \vdash^{\mathrm{ML}} e_0 : u_1 \to \cdots u_n \to u_0, \; A' \vdash^{\mathrm{ML}} e_1 : u_1, \cdots, A' \vdash^{\mathrm{ML}} e_n : u_n}{A' \vdash^{\mathrm{ML}} e_0 \mathbin{@}^s_n < e_1, \cdots, e_n > : u_0}$$

$$(\text{let}) \quad \frac{A' \vdash^{\mathrm{ML}} e_1 : us_1, \; A'[x : us_1] \vdash^{\mathrm{ML}} e_2 : u_2}{A' \vdash^{\mathrm{ML}} \texttt{let}\; x = e_1 \;\texttt{in}\; e_2 : u_2}$$

$$(\text{rec}) \quad \frac{A'[f : u] \vdash^{\mathrm{ML}} \texttt{fn}\; x \Rightarrow e : u}{A' \vdash^{\mathrm{ML}} \texttt{rec}\; f\; x \Rightarrow e : u}$$

$$(\text{if}) \quad \frac{A' \vdash^{\mathrm{ML}} e_0 : \texttt{bool}, \; A' \vdash^{\mathrm{ML}} e_1 : u, \; A' \vdash^{\mathrm{ML}} e_2 : u}{A' \vdash^{\mathrm{ML}} \texttt{if}\; e_0 \;\texttt{then}\; e_1 \;\texttt{else}\; e_2 : u}$$

$$(\text{ins}) \quad \frac{A' \vdash^{\mathrm{ML}} e : \forall \vec{\alpha}.u}{A' \vdash^{\mathrm{ML}} e : R\,u} \qquad \text{if } Dom(R) \subseteq \{\vec{\alpha}\}$$

$$(\text{gen}) \quad \frac{A' \vdash^{\mathrm{ML}} e : u}{A' \vdash^{\mathrm{ML}} e : \forall \vec{\alpha}.u} \qquad \text{if } FV(A') \cap \{\vec{\alpha}\} = \emptyset$$

Figure 2.9: Our variant of the ML type inference system

defined on sequential constants only and let it behave as in Fig. 2.4; and let $\epsilon(A) = A'$.

- If $A' \vdash_n^{\mathrm{ML}} e : u$ then there exists $\beta$-sequential type $t$ with $\epsilon(t) = u$ such that $C_\beta, A \vdash_n e : t \,\&\, \beta$.

- If $C, A \vdash e : t \,\&\, b$ where $C$ contains no type constraints then there exists an ML type $u$ with $\epsilon(t) = u$ such that $A' \vdash^{\mathrm{ML}} e : u$.

**Proof** See Appendix A. □

So if e.g. $A' \vdash^{\mathrm{ML}} e : \mathtt{int} \rightarrow \mathtt{int}$ then we may expect that also

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash e : \mathtt{int} \rightarrow^{\beta} \mathtt{int} \,\&\, \beta.$$

**Remark** We restrict our attention to expressions in *Exp*, as the (first half of the) theorem does not hold in general for *EExp*: consider e.g. the expression `cons` 3 which is typeable in ML but not in our system. □

## 2.9 Properties of the Arrow Relation

For the subsequent development it is crucial that if $C \vdash b \subseteq b'$ then $FV(b)^{C\downarrow} \subseteq FV(b')^{C\downarrow}$, provided $C$ is sufficiently "well-behaved".[8] In order to prove this result it is convenient to consider *forward derivations* only; we shall write $C \vdash_{fw} g_1 \subseteq g_2$ if $C \vdash g_1 \subseteq g_2$ can be derived from Figs. 2.6 and 2.7 and 2.8 without using the rules labelled (bw). We have followed a non-standard approach by incorporating these explicit rules for decomposition; we shall see that for suitable $C$ these rules do not add to the power of the system.

In general it does not hold that $C \vdash g_1 \subseteq g_2$ implies $\overline{C} \vdash_{fw} g_1 \subseteq g_2$ even if $C$ (and hence $\overline{C}$) is well-formed. To see this, let

$$C = \{\alpha \subseteq \alpha',\ \mathtt{int\ event}\ \beta \subseteq \alpha\ \mathtt{list},\ \alpha'\ \mathtt{list} \subseteq \mathtt{int\ event}\ \beta'\}$$

[8]The result is formalised as Lemma 2.29 and is needed, together with Lemma 2.33, to establish the cases for (gen) in the proofs of Lemma 3.24, essential for semantic soundness, and Theorem 5.18, demonstrating the completeness of our reconstruction algorithm.

thus $C$ is well-formed and $\overline{C} = C$. As $C \vdash \alpha$ `list` $\subseteq \alpha'$ `list` transitivity yields $C \vdash$ `int event` $\beta \subseteq$ `int event` $\beta'$ so by (bw) we get $C \vdash (\beta \subseteq \beta')$; but it is clearly impossible to derive $C \vdash_{fw} (\beta \subseteq \beta')$. This example motivates the following definition:

**Definition 2.26** We say that there is a *mismatch* between two non-variable types $t_1$ and $t_2$ if their top-level type constructors are different.

We say that a constraint set $C$ is *consistent* if for all $t_1$,$t_2$ where there is a mismatch between $t_1$ and $t_2$ it is impossible to derive $C \vdash t_1 \subseteq t_2$. □

The notion of consistency is what is needed in order to dispense with the explicit decomposition rules:

**Lemma 2.27** Consider the inference rules in Figs. 2.6 and 2.7 and 2.8, where we assume that $C$ is consistent and backwards closed (i.e. $\overline{C} = C$). Then for all rules labelled (bw) the following holds: if the premise has a forward derivation, then also the conclusion has a forward derivation.

**Proof** We will show that if $C \vdash_{fw} t$ `event` $\beta \subseteq t'$ `event` $\beta'$ then $C \vdash_{fw} t \subseteq t'$ and $C \vdash_{fw} \beta \subseteq \beta'$; the other cases are similar.

It is easy to see that there exists $n \geq 0$ and $t'_0 \cdots t'_n$ with $t'_0 = t$ `event` $\beta$ and $t'_n = t'$ `event` $\beta'$, such that for all $i \in \{0 \cdots n \Leftrightarrow 1\}$ we have $C \vdash_{fw} t'_i \subseteq t'_{i+1}$ where the last rule applied is neither (refl) nor (trans).

As $C$ is consistent by assumption, each $t'_i$ must be either a variable or an `event`-type. We shall enumerate the latter kind of indices: let $m$ be the number of $i$'s in $\{0 \cdots n\}$ with $t'_i$ a `event`-type; then $m \geq 1$ and there exists a strictly monotone sequence $i_1 \cdots i_m$ (with $i_1 = 0$ and $i_m = n$), types $t_1 \cdots t_m$, and behaviour variables $\beta_1 \cdots \beta_m$, such that for all $j \in \{1 \cdots m\}$ we have $t'_{i_j} = t_j$ `event` $\beta_j$. As we clearly have $t_1 = t$, $\beta_1 = \beta$, $t_m = t'$ and $\beta_m = \beta'$, our task can be accomplished by showing that for all $j \in \{1 \cdots m \Leftrightarrow 1\}$ it holds that $C \vdash_{fw} t_j \subseteq t_{j+1}$ and $C \vdash_{fw} \beta_j \subseteq \beta_{j+1}$.

For a given $j$ we distinguish between two cases:

*(i)* If $i_{j+1} = i_j + 1$ the situation is that $C \vdash_{fw} t_j$ `event` $\beta_j \subseteq t_{j+1}$ `event` $\beta_{j+1}$ where the last rule applied is neither (refl) nor (trans); as the rules labelled (bw) are not permitted the last rule applied must be either (`event`) or (axiom). In the former case the claim follows directly; in the latter case the claim follows from $C$ being backwards closed.

*(ii)* Otherwise the situation is that there exists $\alpha_1 \cdots \alpha_p$ with $p \geq 1$ such that $C \vdash_{fw} t_j$ `event` $\beta_j \subseteq \alpha_1$, $C \vdash_{fw} \alpha_k \subseteq \alpha_{k+1}$ for all $k \in \{1 \cdots p \Leftrightarrow 1\}$, and $C \vdash_{fw} \alpha_p \subseteq t_{j+1}$ `event` $\beta_{j+1}$, where the last rule applied in all these inferences is neither (refl) nor (trans). As the rules labelled (bw) are not permitted we infer that the last rule applied in all those inferences is (axiom). The claim now follows from $C$ being backwards closed. □

**Corollary 2.28** Assume that $C$ is consistent and that $C \vdash (g_1 \subseteq g_2)$. Then $\overline{C} \vdash_{fw} (g_1 \subseteq g_2)$.

**Proof** Clearly $\overline{C}$ is consistent and $\overline{C} \vdash (g_1 \subseteq g_2)$; the result now follows by induction in the latter inference using Lemma 2.27. □

We are now ready for the main result, as promised in the beginning of this section:

**Lemma 2.29** Assume that $C \vdash b \subseteq b'$ with $C$ well-formed and consistent. Then for all $\gamma \in FV(b)$ there exists $\gamma' \in FV(b')$ such that $C \vdash \gamma \leftarrow^* \gamma'$. (That is, $FV(b)^{C\downarrow} \subseteq FV(b')^{C\downarrow}$.)

**Proof** By Corollary 2.28 we have $\overline{C} \vdash_{fw} b \subseteq b'$; we perform induction in this derivation. Most cases are straightforward; we only spell out the case (axiom) in some detail: suppose that $\overline{C} \vdash_{fw} b \subseteq b'$ because $(b \subseteq b') \in \overline{C}$. As the constraint set $\overline{C}$ is well-formed (Fact 2.11) we infer that $b'$ is a variable $\beta'$. Let $\gamma \in FV(b)$, then the desired relation $C \vdash \gamma \leftarrow \beta'$ holds by the definition of $\leftarrow$. □

Next some results showing that the arrow relation is in some sense closed under substitution and entailment:

**Lemma 2.30** Suppose $C \vdash \gamma \leftarrow \beta$. Let $S$ be a substitution; then for all $\gamma' \in FV(S\,\gamma)$ it holds that $S\,C \vdash \gamma' \leftarrow S\,\beta$.

**Proof** There exists $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in \overline{C}$; that is (cf. Definition 2.7) there is a derivation $C \vdash_{dc} (b \subseteq \beta)$. By Lemma 2.18 we infer that there also is a derivation $S\,C \vdash_{dc} (S\,b \subseteq S\,\beta)$, implying that $(S\,b \subseteq S\,\beta) \in \overline{S\,C}$. For all $\gamma' \in FV(S\,\gamma)$ we also have $\gamma' \in FV(S\,b)$ and therefore $S\,C \vdash \gamma' \leftarrow S\,\beta$ holds as desired. □

**Corollary 2.31** Suppose $C \vdash \gamma_1 \leftarrow^* \gamma_2$. Let $S$ be a substitution; then for all $\gamma_1' \in FV(S\,\gamma_1)$ there exists $\gamma_2' \in FV(S\,\gamma_2)$ such that $S\,C \vdash \gamma_1' \leftarrow^* \gamma_2'$.

**Proof** Induction in the length of the sequence $C \vdash \gamma_1 \leftarrow^* \gamma_2$; if the length is zero then $\gamma_1 = \gamma_2$ and the claim is trivial. So assume that there exists $\beta$ such that $C \vdash \gamma_1 \leftarrow \beta$ and $C \vdash \beta \leftarrow^* \gamma_2$ (by a shorter derivation). Given $\gamma_1' \in FV(S\,\gamma_1)$ we by Lemma 2.30 infer that $S\,C \vdash \gamma_1' \leftarrow S\,\beta$; and the induction hypothesis tells us that there exists $\gamma_2' \in FV(S\,\gamma_2)$ such that $S\,C \vdash S\,\beta \leftarrow^* \gamma_2'$. This yields the claim. □

**Lemma 2.32** Suppose $C \vdash \gamma \leftarrow \beta$ and that $C' \vdash C$ with $C'$ well-formed and consistent; then $C' \vdash \gamma \leftarrow^* \beta$.

**Proof** There exists $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in \overline{C}$; thus it holds that $C' \vdash b \subseteq \beta$. The claim now follows from Lemma 2.29. □

Finally a result which proves useful later on:

**Lemma 2.33** Suppose the type scheme $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is well-formed (cf. Definition 2.12) and that $C$ is well-formed with $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C) = \emptyset$. Then $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{C \cup C_0 \uparrow} = \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$.

**Proof** Let $C \cup C_0 \vdash \gamma \leftarrow \beta$ with $\gamma \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$; our task is to show that $\beta \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$. There exists $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in \overline{C \cup C_0}$, leaving us with 3 cases:

- $(b \subseteq \beta) \in C^b$: this is impossible as $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C) = \emptyset$.
- $(b \subseteq \beta) \in {C_0}^b$: then $C_0 \vdash \gamma \leftarrow \beta$ so as $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{C_0\uparrow}$ (since $ts$ is well-formed) we infer $\beta \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ as desired.
- $(b \subseteq \beta) \in \overline{C^t \cup {C_0}^t}$: $b$ must be a behaviour variable and thus equal $\gamma$, that is $\gamma \in \vec{\beta}$. As $ts$ is well-formed $\gamma \notin FV({C_0}^t)$ and by assumption $\gamma \notin FV(C^t)$; showing that also this case is impossible.

□

# Chapter 3

# The Dynamic Semantics

In this chapter we define a dynamic semantics which employs one system for the sequential components (Sect. 3.1) and another for the concurrent components (Sect. 3.2). Next (Sect. 3.3) we extend the repertoire of techniques (from Chap. 2) for normalising and manipulating the inference trees of the annotated type and effect system. Finally, we show that this system is indeed semantically sound with respect to the dynamic semantics: we establish a sequential subject reduction result (Theorem 3.23) as a preparation for a concurrent subject reduction result (Theorem 3.28) which shows that the concurrent transition relation "preserves types" and "decreases behaviours", and which also demonstrates that the actions performed by the system are in a certain sense as "predicted" by the behaviour information. Moreover, in Sect. 3.5 we demonstrate (informally) that it is not possible to assign a type to the "error configurations" which have been characterised in Proposition 3.9.

It is a crucial feature of the soundness theorem that it only considers *channel environments*, where $A$ is a channel environment if the identifiers in $Dom(A)$ are all channel identifiers[1] and for each $ch \in Dom(A)$ that $A(ch)$ takes the form $t$ `chan` $\rho$. The "initial environment" (where only constants are in the domain) is a channel environment, and we shall see that the concurrent soundness result 3.28 guarantees that the assumption is maintained; thus our restriction seems to be a benign one. To see that it is actually necessary

[1] We assume that the set of identifiers contains the set of channel identifiers as an infinite subset; we use $ch$ to range over such identifiers.

| $F_s$ | $e$ | $\delta(F_s\!<\!e\!>)$ |
|---|---|---|
| fst | pair$<e_1, e_2>$ | $e_1$ |
| snd | pair$<e_1, e_2>$ | $e_2$ |
| hd | cons$<e_1, e_2>$ | $e_1$ |
| hd | nil | hd$<$nil$>$ |
| tl | cons$<e_1, e_2>$ | $e_2$ |
| tl | nil | tl$<$nil$>$ |
| null | nil | true |
| null | cons$<e_1, e_2>$ | false |
| + | pair$<n_1, n_2>$ | $n$ where $n = n_1 + n_2$ |
| $\vdots$ | $\vdots$ | |
| / | pair$<n, 0>$ | /$<$pair$<n, 0>>$ |

Figure 3.1: The evaluation function $\delta$

to impose the condition, note that otherwise the type of the channel would be polymorphic and the sender and receiver of a transmitted value would then be allowed to disagree on its type; this is exactly where type insecurities would creep in.

## 3.1 The Sequential Semantics

We are going to define a small-step semantics for the sequential part of the language. Transitions take the form $e \to e'$ where $e$ and $e'$ are expressions in *Exp* that are *essentially closed*: this means that all free identifiers are channel identifiers (created by previous channel allocations).

We first stipulate the semantics of the sequential base functions $F_s$ (such as + or fst) by means of an "evaluation function" $\delta$:

**Definition 3.1** The function $\delta$ is a partial mapping from expressions of the form $F_s < e >$ into expressions (preserving the property of being essentially closed): It is defined by the (incomplete) Figure 3.1; notice that we encode "runtime errors" such as hd (nil) as loops whereas e.g. hd (7) is undefined. □

We next introduce the notion of *weakly evaluated expressions* ($w \in$ *WExp*) that are the "terminal configurations" of the sequential semantics:

**Definition 3.2** An expression $w$ is a *weakly evaluated expression* provided that either

- $w$ is a channel identifier $ch$; or
- $w$ is a function abstraction $\texttt{fn}\ x \Rightarrow e$; or
- $w$ is of the form $C^n < w_1, \cdots, w_n >$ where $n \geq 0$, where $w_1, \cdots, w_n$ are weakly evaluated expressions, and where $C^n$ is a $n$-ary constructor (sequential or non-sequential). □

To formalise the call-by-value evaluation strategy we shall employ the notion of *evaluation context*:

**Definition 3.3** Evaluation contexts $E$ take the form

$$\begin{array}{lll} E & ::= & [\,] \mid E\ e \mid w\ E \\ & \mid & \texttt{let}\ x = E\ \texttt{in}\ e \mid \texttt{if}\ E\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\ & \mid & F\!<\!E\!>\mid C^n\!<\!w_1, \cdots, w_{i-1}, E, e_{i+1}, \cdots, e_n\!> \end{array}$$

Notice that $E$ is a context with exactly one hole in it, and that this hole is not inside the scope of any defining occurrence of a program identifier. We write $E[e]$ for the expression that has the hole in $E$ replaced by $e$, and similarly $E[E']$ for the evalution context that results by replacing the hole in $E$ with $E'$. The following (rather obvious) fact is proved in Appendix B:

**Fact 3.4** $(E_1[E_2])[e] = E_1[E_2[e]]$. □

Now we are ready for:

**Definition 3.5** *Sequential Evaluation*

The sequential transition relation $\rightarrow$ is defined by

> $E[e] \rightarrow E[e']$ provided $e \rightharpoonup e'$ holds according to the following definition:

$$
\begin{array}{rlcl}
\text{(apply)} & (\texttt{fn}\ x \Rightarrow e)\ w & \rightharpoonup & e[w/x] \\
\text{(delta)} & F_s\!<\!w\!> & \rightharpoonup & e' \text{ if } e' = \delta(F_s\!<\!w\!>) \\
\text{(let)} & \texttt{let}\ x = w\ \texttt{in}\ e & \rightharpoonup & e[w/x] \\
\text{(rec)} & \texttt{rec}\ f\ x \Rightarrow e & \rightharpoonup & (\texttt{fn}\ x \Rightarrow e)[(\texttt{rec}\ f\ x \Rightarrow e)/f] \\
\text{(branch)} & \texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \rightharpoonup & \begin{cases} e_1 & \text{if } w = \texttt{true} \\ e_2 & \text{if } w = \texttt{false} \end{cases}
\end{array}
$$

**Fact 3.6** If $e \to e'$ with $e$ essentially closed then also $e'$ is essentially closed.

Observe that $e_1\ e_2 \to e'$ holds iff either *(i)* $e_1\ e_2 \rightharpoonup e'$, or *(ii)* there exists $e'_1$ such that $e_1 \to e'_1$ and $e' = e'_1\ e_2$, or *(iii)* there exists $e'_2$ such that $e_2 \to e'_2$ and $e' = e_1\ e'_2$ (in which case $e_1$ is a weakly evaluated expression). Further observe that $\texttt{let}\ x = e_1\ \texttt{in}\ e_2 \to e'$ holds iff either *(i)* $\texttt{let}\ x = e_1\ \texttt{in}\ e_2 \rightharpoonup e'$, or *(ii)* there exists $e'_1$ such that $e_1 \to e'_1$ and $e' = \texttt{let}\ x = e'_1\ \texttt{in}\ e_2$; and observe that $\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \to e'$ holds iff either *(i)* $\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \rightharpoonup e'$, or *(ii)* there exists $e'_0$ such that $e_0 \to e'_0$ and $e' = \texttt{if}\ e'_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. Finally observe that $F\!<\!e_1\!>\to e'$ holds iff either *(i)* $e' = \delta(F\!<\!e\!>)$ (in which case $F$ is sequential), or *(ii)* there exists $e'_1$ such that $e_1 \to e'_1$ and $e' = F\!<\!e'_1\!>$; and observe that $C^n\!<\!e_1, \cdots, e_n\!>\to e'$ holds if there exists $i \in \{1 \cdots n\}$ and $e'_i$ such that $e_i \to e'_i$ and $e' = C^n\!<\!e_1, \cdots, e'_i, \cdots, e_n\!>$ (in which case $e_1 \cdots e_{i-1}$ are weakly evaluated expressions).

As expected we have:

**Fact 3.7** If $w$ is a weakly evaluated expression then $w \not\to$.

**Proof** It is easy to see that $w \not\rightharpoonup$; the result then follows by an easy induction on $w$. □

We shall say that an essentially closed expression $e$ is *exhausted* if it is not weakly evaluated and yet $e \not\to$. We shall say that an exhausted expression $e$ is *top-level exhausted* if it cannot be written on the form $e = E[e']$ with $E \neq [\,]$ and with $e'$ exhausted. It is easy to see (using Fact 3.4) that for any exhausted expression $e$ there exists $E$ and top-level exhausted $e'$ such that $e = E[e']$.

**Fact 3.8** Suppose that $e$ is top-level exhausted; then either

- $e =$ `if` $w$ `then` $e_1$ `else` $e_2$ with $w \notin \{\texttt{true}, \texttt{false}\}$; or
- $e = ch\ w$ with $ch$ a channel identifier; or
- $e = (C^n\!<\!w_1, \cdots, w_n\!>)\ w$; or
- $e = F_c\!<\!w\!>$; or
- $e = F_s\!<\!w\!>$ with $\delta(e)$ undefined.

**Proof** We perform a case analysis on the essentially closed expression $e$. If $e$ is a channel identifier or an abstraction then $e$ is weakly evaluated and hence not exhausted. If $e$ is of the form `rec` $f\ x \Rightarrow e$, then $e \rightharpoonup \cdots$ and hence $e$ is not exhausted.

If $e$ is of the form `let` $x = e_1$ `in` $e_2$ then $e_1$ is essentially closed and $e_1 \not\rightarrow$ (as otherwise $e \rightarrow$) but $e_1$ is not exhausted (as $e$ is top-level exhausted). Hence we conclude that $e_1$ is weakly evaluated, but this is a contradiction since then $e \rightharpoonup \cdots$.

If $e$ is of the form `if` $e_0$ `then` $e_1$ `else` $e_2$ then $e_0$ is essentially closed and $e_0 \not\rightarrow$ (as otherwise $e \rightarrow$) but $e_0$ is not exhausted (as $e$ is top-level exhausted). Hence we conclude that $e_0$ is weakly evaluated; and this yields the claim since if $e_0 = \texttt{true}$ or $e_0 = \texttt{false}$ then $e \rightharpoonup \cdots$.

If $e$ is of the form $e_1\ e_2$ we infer (using the same technique as in the above two cases) that $e_1$ is a weakly evaluated expression $w_1$ and subsequently that $e_2$ is a weakly evaluated expression $w_2$. This yields the claim since if $w_1$ is an abstraction then $e \rightharpoonup \cdots$.

If $e$ is of the form $F < e_1 >$ we infer (in the usual way) that $e_1$ is a weakly evaluated expression $w_1$; this yields the claim since if $F$ is sequential and $\delta(F\!<\!w_1\!>)$ is defined then $e \rightharpoonup \cdots$.

If $e$ is of the form $C^n < e_1, \cdots, e_n >$ we infer (by subsequent applications of the by now familiar reasoning technique) that $e_1, \cdots, e_n$ are weakly evaluated expressions; thus also $e$ is weakly evaluated and hence not exhausted. □

From the preceding results we get:

**Proposition 3.9** Suppose that $e$ is essentially closed and that $e \rightarrow^* e' \not\rightarrow$. Then either

1. $e'$ is a weakly evaluated expression; or

2. $e'$ is of the form $E[F_c\!<\!w\!>]$; or

3. $e'$ is of the form $E[e'']$ with either

   - $e'' = \texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ with $w \notin \{\texttt{true}, \texttt{false}\}$; or
   - $e'' = ch\ w$ with $ch$ a channel identifier; or
   - $e'' = (C^n\!<\!w_1, \cdots, w_n\!>)\ w$; or
   - $e'' = F_s\!<\!w\!>$ with $\delta(e'')$ undefined. □

The configurations listed in case 3 can be thought of as error configurations, whereas in Section 3.2 we shall see that case 2 corresponds to a process that may be able to perform a concurrent action.

**Fact 3.10** The rewriting relation $\rightarrow$ is deterministic.

**Proof** We perform induction on $e$ to show that if $e \rightarrow e'$ and $e \rightarrow e''$ then $e' = e''$. If $e$ is an identifier or a function abstraction then $e \not\rightarrow$ and if $e$ is of the form $\texttt{rec}\ f\ x \Rightarrow e$ determinism is obvious.

If $e$ is of the form $\texttt{let}\ x = w\ \texttt{in}\ e_2$ the claim follows from $w \not\rightarrow$. If $e$ is of the form $\texttt{let}\ x = e_1\ \texttt{in}\ e_2$ with $e_1$ not a weakly evaluated expression then $e'$ takes the form $\texttt{let}\ x = e_1'\ \texttt{in}\ e_2$ where $e_1 \rightarrow e_1'$ and by the induction hypothesis this $e_1'$ is unique.

If $e$ is of the form $\texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ the claim follows from $w \not\rightarrow$. If $e$ is of the form $\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ with $e_0$ not a weakly evaluated expression then $e'$ takes the form $\texttt{if}\ e_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ where $e_0 \rightarrow e_0'$ and by the induction hypothesis this $e_0'$ is unique.

If $e$ is of the form $F\!<\!w\!>$ the claim follows from $w \not\rightarrow$ and from $\delta$ being a function. If $e$ is of the form $F\!<\!e_1\!>$ with $e_1$ not a weakly evaluated expression then $e'$ takes the form $F\!<\!e_1'\!>$ where $e_1 \rightarrow e_1'$ and by the induction hypothesis this $e_1'$ is unique.

If $e$ is of the form $C^n < w_1, \cdots, w_n >$ the claim follows from $e \not\rightarrow$. If $e$ is of the form $C^n\!<\!w_1, \cdots, w_{i-1}, e_i, \cdots, e_n\!>$ $(i \leq n)$ then $e'$ takes the form $C^n\!<\!w_1, \cdots, w_{i-1}, e_i', \cdots, e_n\!>$ where $e_i \rightarrow e_i'$ and by the induction hypothesis this $e_i'$ is unique.

We are left with the case $e = e_1\ e_2$. First suppose that $e_1$ is not weakly evaluated. Then $e \not\rightharpoonup$ so we infer that $e'$ takes the form $e'_1\ e_2$ where $e_1 \rightarrow e'_1$ and by the induction hypothesis this $e'_1$ is unique.

Next suppose that $e = w_1\ e_2$ with $e_2$ not weakly evaluated. Then $e \not\rightharpoonup$ so as $w_1 \not\rightarrow$ we infer that $e'$ takes the form $w_1\ e'_2$ where $e_2 \rightarrow e'_2$ and by the induction hypothesis this $e'_2$ is unique.

Finally assume that $e = w_1\ w_2$. Then $w_1 \not\rightarrow$ and $w_2 \not\rightarrow$ so it must hold that $e \rightharpoonup e'$. Thus $w_1$ is a function abstraction and then $e'$ is clearly unique. □

## 3.2 The Concurrent Semantics

Next we are going to define a small-step semantics for the concurrent part of the language. Transitions take the form $PP \overset{sa}{\Leftrightarrow\!\!\!\rightarrow} PP'$, where $PP$ as well as $PP'$ is a *process pool* which is a finite mapping from process identifiers $p$ into essentially closed expressions $\in$ *Exp*, and where $sa$ is a label describing what kind of semantic action is taken.

**Definition 3.11** *Concurrent Evaluation*

The concurrent transition relation $\overset{sa}{\Leftrightarrow\!\!\!\rightarrow}$ is defined by:

$$PP[p:e] \quad \overset{\texttt{seq}}{\Leftrightarrow\!\!\!\rightarrow} \quad PP[p:e'] \qquad \text{if } e \rightarrow e'$$

$$PP[p:E[\texttt{channel}^l\texttt{<()>}]] \quad \overset{p\ \texttt{chan}^l\ ch}{\Leftrightarrow\!\!\!\rightarrow} \quad PP[p:E[ch]]$$
$$\text{if the channel identifier } ch \text{ is not in } PP \text{ or } E$$

$$PP[p:E[\texttt{spawn<}w\texttt{>}]] \quad \overset{p\ \texttt{spawn}\ p'}{\Leftrightarrow\!\!\!\rightarrow} \quad PP[p:E[\texttt{()}]][p':w\ \texttt{()}]$$
$$\text{if } p' \notin Dom(PP) \cup \{p\}$$

$$PP[p_1:E_1[\texttt{sync<}e_1\texttt{>}]][p_2:E_2[\texttt{sync<}e_2\texttt{>}]] \quad \overset{p_1,p_2\ \texttt{comm}\ ch}{\Leftrightarrow\!\!\!\rightarrow} \quad PP[p_1:E_1[w]][p_2:E_2[w]]$$
$$\text{if } e_1 = \texttt{transmit<pair<}ch, w\texttt{>>} \text{ and } e_2 = \texttt{receive<}ch\texttt{>} \text{ and } p_1 \neq p_2$$

## 3.3 Reasoning about Proof Trees

In this section we present some auxiliary results which will eventually enable us to show that if there is a typing for $e$ and if $e$ gets "rewritten" into $e'$ (sequentially or concurrently) then we can construct a typing for $e'$.

A common pattern will be that we have some judgement $C, A \vdash E[e] : \sigma \,\&\, b$, but we want to reason about the typing of $e$ rather than that of $E[e]$. To this end we need to be precise about what it means for a judgement to occur "at the address indicated by the hole in $E$"; motivated by the translation in Fig. 2.3 (from *Exp* to *EExp*) we stipulate:

**Definition 3.12** The judgement $jdg' = (C', A' \vdash e' : \sigma' \,\&\, b')$ occurs at $E$ (with depth $n$) in the inference tree for the judgement $jdg = (C, A \vdash e : \sigma \,\&\, b)$, provided that *either*

$$jdg = jdg' \text{ and } E = [\,] \text{ (and } n = 0)$$

*or* there exists a judgement $jdg''$ and an evaluation context $E''$ such that $jdg'$ occurs at $E''$ (with depth $n \Leftrightarrow 1$) in the inference tree for $jdg''$, and such that the last rule applied in the inference tree for $jdg$ is *either*

- (sub), (ins), or (gen), with $jdg''$ as premise and with $E = E''$; *or*
- (app), with $jdg''$ as leftmost premise and with $E = E''\ e_2$ where $e$ is of the form $e''\ e_2$; *or*
- (app), with $jdg''$ as rightmost premise and with $E = w_1\ E''$ where $e$ is of the form $w_1\ e''$; *or*
- (app), with $jdg''$ as rightmost premise and with $E = F_c < E'' >$ where $e$ is of the form $F_c < e'' >$; *or*
- (sapp), with $jdg''$ as rightmost premise and with $E = F_s < E'' >$ where $e$ is of the form $F_s < e'' >$; *or*
- (sapp), with $jdg''$ as premise no. $i + 1$ and with $E = C^n < w_1, \cdots, w_{i-1}, E'', e_{i+1}, \cdots, e_n > \ (i \leq n)$ where $e$ is of from $C^n < w_1, \cdots, w_{i-1}, e'', e_{i+1}, \cdots, e_n >$; *or*

- (let), with $jdg''$ as leftmost premise and with $E = \texttt{let}\ x = E''\ \texttt{in}\ e_2$ where $e$ is of the form $\texttt{let}\ x = e''\ \texttt{in}\ e_2$; *or*

- (if), with $jdg''$ as leftmost premise and with $E = \texttt{if}\ E''\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ where $e$ is of the form $\texttt{if}\ e''\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. □

This is well-defined in the size of the inference tree for $jdg$. As expected we have the following results, the latter to be proved in Appendix B:

**Fact 3.13** Suppose that $C', A' \vdash e' : \sigma' \,\&\, b'$ occurs at $E$ in the inference tree for $C, A \vdash e : \sigma \,\&\, b$; then $e = E[e']$. □

**Fact 3.14** Given $jdg = (C, A \vdash E[e] : \sigma \,\&\, b)$; then there exists (at least one) judgement $jdg'$ of the form $C', A' \vdash e : \sigma' \,\&\, b'$ such that $jdg'$ occurs at $E$ in the inference tree for $jdg$. If $jdg$ is normalised we can assume that $jdg'$ is normalised. □

Some of the subsequent proofs will be by induction in the depth of a judgement in an inference tree; for this purpose the following result is convenient:

**Fact 3.15** Suppose the judgement $jdg'$ occurs at $E$ with depth $n$ in the inference tree for $jdg$, where $n \geq 2$. Then there exists a judgement $jdg''$ and evaluation contexts $E_1$ and $E_2$ such that

> $jdg'$ occurs at $E_1$ with depth $< n$ in the inference tree for $jdg''$; and
> $jdg''$ occurs at $E_2$ with depth $< n$ in the inference tree for $jdg$; and
> $E = E_2[E_1]$.

Moreover, if $jdg$ is normalised we can assume that also $jdg''$ is normalised.

**Proof** We can clearly use $jdg''$ as in Definition 3.12 (and choose $E_1$ as $E''$); notice that if $jdg$ is normalised then (as $n \geq 2$) the last rule applied cannot be (ins). □

Having set up the necessary machinery we are now ready for the first result, which states that "equivalent" expressions may be substituted for each other:

**Fact 3.16** Suppose the judgement $C', A' \vdash_n e : \sigma' \,\&\, b'$ occurs at $E$ in the inference tree of $C, A \vdash_n E[e] : \sigma \,\&\, b$. If $e_n$ is such that $C', A' \vdash_n e_n : \sigma' \,\&\, b'$ then also $C, A \vdash_n E[e_n] : \sigma \,\&\, b$. □

It proves useful to know something about the relationship between the root of an inference tree and the interior nodes of the tree:

**Lemma 3.17** Suppose the judgement $C', A' \vdash e' : \sigma' \,\&\, b'$ occurs at $E$ in the inference tree of $C, A \vdash e : \sigma \,\&\, b$. Then

- $A' = A$;
- if $C$ is well-formed then also $C'$ is well-formed;
- if $C$ is consistent then also $C'$ is consistent.

**Proof** See Appendix B. □

The next two lemmas tell us something about the relationship between the type of an expression $c{<}e_1, \cdots, e_n{>}$, the type of $c$, and the type of each $e_i$.

**Lemma 3.18** Suppose that with $c \in C^n$, or $c \in F_s$ and $n = 1$, we have

$$C, A \vdash_n c{<}e_1, \cdots, e_n{>} : t \,\&\, b$$

and that $A(c)$ is of the form (cf. Fact 2.16)

$$\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t'_1 \rightarrow \cdots t'_n \rightarrow t'.$$

Then there exists $S$, $t_1 \cdots t_n$, and $b_1 \cdots b_n$, such that

$$Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \text{ and } C \vdash S\,C_0 \text{ and } C \vdash S\,t' \subseteq t;$$

$$\text{for all } i \in \{1 \cdots n\}\text{: } C, A \vdash_n e_i : t_i \,\&\, b_i \text{ and } C \vdash t_i \subseteq S\,t'_i;$$

$$C \vdash b_1; \cdots; b_n \subseteq b.$$

Similarly, if $A(c) = t'_1 \rightarrow \cdots t'_n \rightarrow t'$ in which case $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$ and $C_0 = \emptyset$ (so we have $S = \mathrm{Id}$).

**Proof** The situation must be

$$\frac{C, A \vdash_n c : t_1 \to \cdots t_n \to t_0 \,\&\, b_0 \cdots C, A \vdash_n e_i : t_i \,\&\, b_i \cdots}{C, A \vdash_n c\!<\!e_1, \cdots, e_n\!> : t \,\&\, b}\text{(sapp),(sub)}^*$$

with $C \vdash t_0 \subseteq t$ and $C \vdash b_0; b_1; \cdots; b_n \subseteq b$. The leftmost premise has a derivation tree

$$\cfrac{\cfrac{C, A \vdash c : A(c) \,\&\, \varepsilon}{C, A \vdash_n c : S\,t_1' \to \cdots S\,t_n' \to S\,t' \,\&\, \varepsilon}\text{ (ins)}}{C, A \vdash_n c : t_1 \to \cdots t_n \to t_0 \,\&\, b_0}\text{ (sub)}^*$$

where the instance substitution $S$ satisfies $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \vdash S\,C_0$. All the claims now follow immediately. □

**Lemma 3.19** Suppose that we have

$$C, A \vdash_n F_c\!<\!e_1\!> : t \,\&\, b$$

and that $A(F_c)$ is of the form

$$\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_1' \to^{\beta'} t'.$$

Then there exists $S$, $t_1$, and $b_1$, such that

$Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \vdash S\,C_0$ and $C \vdash S\,t' \subseteq t$;

$C, A \vdash_n e_1 : t_1 \,\&\, b_1$ and $C \vdash t_1 \subseteq S\,t_1'$ and $C \vdash b_1; S\,\beta' \subseteq b$.

**Proof** The situation must be

$$\frac{C, A \vdash_n F_c : t_1 \to^{\beta} t_0 \,\&\, b_0 \qquad C, A \vdash_n e_1 : t_1 \,\&\, b_1}{C, A \vdash_n F_c\!<\!e_1\!> : t \,\&\, b}\text{(app),(sub)}^*$$

with $C \vdash t_0 \subseteq t$ and $C \vdash b_0; b_1; \beta \subseteq b$. The leftmost premise has a derivation tree

$$\cfrac{\cfrac{C, A \vdash F_c : A(F_c) \,\&\, \varepsilon}{C, A \vdash_n F_c : S\,t_1' \to^{S\,\beta'} S\,t' \,\&\, \varepsilon}\text{ (ins)}}{C, A \vdash_n F_c : t_1 \to^{\beta} t_0 \,\&\, b_0}\text{ (sub)}^*$$

where the instance substitution $S$ satisfies $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \vdash S\,C_0$. All the claims now follow immediately; in particular we have $C \vdash b_1; S\,\beta' \equiv \varepsilon; b_1; S\,\beta' \subseteq b_0; b_1; \beta \subseteq b$. □

The following two lemmas, both to be proved in Appendix B, show

- that we can replace variables by expressions of the same type, provided these expressions have an empty behaviour; and
- that the latter condition can always be obtained for weakly evaluated expressions.

**Lemma 3.20** Suppose that $C, A[x : \sigma'] \vdash_n e : \sigma \,\&\, b$ and that $C, A \vdash_n e' : \sigma' \,\&\, \varepsilon$; then $C, A \vdash_n e[e'/x] : \sigma \,\&\, b$.

**Lemma 3.21** Suppose that $C, A \vdash_n w : \sigma \,\&\, b$; then

- $C \vdash \varepsilon \subseteq b$ and
- $C, A \vdash_n w : \sigma \,\&\, \varepsilon$.

## 3.4 Sequential Soundness

First we shall prove that "top-level" reduction is sound:
**Lemma 3.22** Let $A$ be standard. If $e \rightharpoonup e'$ and

$$C, A \vdash_n e : \sigma \,\&\, b$$

then also

$$C, A \vdash_n e' : \sigma \,\&\, b.$$

**Proof** We perform induction in the proof tree of $C, A \vdash_n e : \sigma \,\&\, b$.

<u>The rule (gen) has been applied:</u> Then the situation is

$$\frac{C \cup C_0, A \vdash_n e : t_0 \,\&\, b}{C, A \vdash_n e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}\ \text{(gen)}$$

and the induction hypothesis yields

$$C \cup C_0, A \vdash_n e' : t_0 \,\&\, b$$

from which we by (gen) arrive at the desired judgement

$$C, A \vdash_n e' : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b.$$

<u>The rule (sub) has been applied:</u> Then the situation is

$$\frac{C, A \vdash_n e : t \,\&\, b}{C, A \vdash_n e : t' \,\&\, b'} \text{ (sub)}$$

and the induction hypothesis yields

$$C, A \vdash_n e' : t \,\&\, b$$

from which we by (sub) arrive at the desired judgement

$$C, A \vdash_n e' : t' \,\&\, b'.$$

<u>Otherwise</u> a "structural" rule has been applied; we now perform case analysis on the transition $\rightharpoonup$:

**The transition (let) has been applied:** Then the situation is

$$\frac{C, A \vdash_n w : ts \,\&\, b_1 \qquad C, A[x : ts] \vdash_n e : t \,\&\, b_2}{C, A \vdash \texttt{let } x = w \texttt{ in } e : t \,\&\, b_1; b_2}$$

and using Lemma 3.21 we have

$$C \vdash \varepsilon \subseteq b_1 \text{ and } C, A \vdash_n w : ts \,\&\, \varepsilon$$

which by Lemma 3.20 can be combined with the second premise of the inference to yield

$$C, A \vdash_n e[w/x] : t \,\&\, b_2$$

and since $C \vdash b_2 \subseteq \varepsilon; b_2 \subseteq b_1; b_2$ we can apply (sub) to get the desired result.

**The transition (rec) has been applied:** Then the situation is

$$\frac{C, A[f : t] \vdash_n \texttt{fn}\ x \Rightarrow e\ :\ t \,\&\, b}{C, A \vdash \texttt{rec}\ f\ x \Rightarrow e\ :\ t \,\&\, b}$$

and using Lemma 3.21 we have

$$C, A[f : t] \vdash_n \texttt{fn}\ x \Rightarrow e\ :\ t \,\&\, \varepsilon$$

so by applying (rec) we get the judgement

$$C, A \vdash_n \texttt{rec}\ f\ x \Rightarrow e\ :\ t \,\&\, \varepsilon$$

which by Lemma 3.20 can be combined with the premise of the inference to yield the desired

$$C, A \vdash_n (\texttt{fn}\ x \Rightarrow e)[(\texttt{rec}\ f\ x \Rightarrow e)/f]\ :\ t \,\&\, b.$$

**The transition (branch) has been applied:** Then the situation is

$$\frac{C, A \vdash_n w\ :\ \texttt{bool} \,\&\, b_0 \qquad C, A \vdash_n e_1\ :\ t \,\&\, b_1 \qquad C, A \vdash_n e_2\ :\ t \,\&\, b_2}{C, A \vdash \texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2\ :\ t \,\&\, b_0; (b_1 + b_2)}$$

and using Lemma 3.21 we have $C \vdash \varepsilon \subseteq b_0$. The claim now follows from the fact that for $i = 1, 2$ we have $C \vdash b_i \subseteq \varepsilon; (b_1 + b_2) \subseteq b_0; (b_1 + b_2)$.

**The transition (apply) has been applied:** Then the situation is

$$\frac{\dfrac{C, A[x : t_2'] \vdash_n e\ :\ t' \,\&\, \beta'}{C, A \vdash_n \texttt{fn}\ x \Rightarrow e\ :\ t_2 \ \to^{\beta}\ t \,\&\, b_1}\ (\text{abs})\ (\text{sub})^* \qquad C, A \vdash_n w\ :\ t_2 \,\&\, b_2}{C, A \vdash (\texttt{fn}\ x \Rightarrow e)\ w\ :\ t \,\&\, (b_1; b_2; \beta)}$$

where $C \vdash \varepsilon \subseteq b_1$ and also $C \vdash t_2' \ \to^{\beta'}\ t' \subseteq t_2 \ \to^{\beta}\ t$ implying that

$$C \vdash t_2 \subseteq t_2' \text{ and } C \vdash \beta' \subseteq \beta \text{ and } C \vdash t' \subseteq t.$$

By Lemma 3.21 followed by an application of (sub) we get

$$C \vdash \varepsilon \subseteq b_2 \text{ and } C, A \vdash_n w\ :\ t_2' \,\&\, \varepsilon$$

which by Lemma 3.20 can be combined with the upmost leftmost premise of the inference to yield

$$C, A \vdash_n e[w/x] : t' \,\&\, \beta'$$

and since $C \vdash t' \subseteq t$ and $C \vdash \beta' \subseteq \beta \subseteq \varepsilon;\varepsilon;\beta \subseteq b_1;b_2;\beta$ we can apply (sub) to get the desired result.

**The transition (delta) has been applied:** The claim then follows from an examination of the figure defining $\delta$; below we shall list a typical case only. In all cases we make use of Lemmas 3.18 and 3.21.

$\underline{e = \texttt{fst}<\texttt{pair}<w_1, w_2>> \textbf{ and } \delta(e) = w_1\textbf{:}}$ Then the situation is that

$$C, A \vdash_n \texttt{fst}<\texttt{pair}<w_1, w_2>> : t \,\&\, b$$

so since $A(\texttt{fst}) = \forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \times \alpha_2 \rightarrow \alpha_1$ Lemma 3.18 tells us that there exists $t_0$, $b_0$ and $S_0$ such that

$$C, A \vdash_n \texttt{pair}<w_1, w_2> : t_0 \,\&\, b_0 \text{ and}$$
$$C \vdash S_0\,\alpha_1 \subseteq t \text{ and } C \vdash t_0 \subseteq S_0\,(\alpha_1 \times \alpha_2) \text{ and } C \vdash b_0 \subseteq b.$$

Since $A(\texttt{pair}) = \forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_1 \times \alpha_2$ Lemma 3.18 tells us that there exists $t_1$, $b_1$, $t_2$, $b_2$ and $S$ such that

$$C, A \vdash_n w_1 : t_1 \,\&\, b_1 \text{ and } C, A \vdash_n w_2 : t_2 \,\&\, b_2;$$
$$C \vdash t_1 \subseteq S\,\alpha_1 \text{ and } C \vdash t_2 \subseteq S\,\alpha_2 \text{ and } C \vdash b_1;b_2 \subseteq b_0;$$
$$C \vdash S\,(\alpha_1 \times \alpha_2) \subseteq t_0$$

and by Lemma 3.21 we infer that $C \vdash \varepsilon \subseteq b_1$ and $C \vdash \varepsilon \subseteq b_2$. Since

$$C \vdash t_1 \times t_2 \subseteq S\,\alpha_1 \times S\,\alpha_2 \subseteq t_0 \subseteq S_0\,\alpha_1 \times S_0\,\alpha_2$$

we deduce that

$$C \vdash t_1 \subseteq S_0\,\alpha_1 \subseteq t$$

and since $C \vdash b_1 \subseteq b_1; \varepsilon \subseteq b_1; b_2 \subseteq b_0 \subseteq b$ we from $C, A \vdash_n w_1 : t_1 \,\&\, b_1$ get the desired judgement

$$C, A \vdash_n w_1 : t \,\&\, b.$$

This completes the proof. □

**Theorem 3.23** *Sequential soundness*
Let $A$ be standard. If $e_1 \to e_2$ and

$$C, A \vdash_n e_1 : \sigma \,\&\, b$$

then also

$$C, A \vdash_n e_2 : \sigma \,\&\, b.$$

**Proof** There exists $E$, $e_1'$ and $e_2'$ such that

$$e_1 = E[e_1'] \text{ and } e_2 = E[e_2'] \text{ and } e_1' \rightharpoonup e_2'.$$

By Fact 3.14 there exists $C'$, $A'$, $\sigma'$ and $b'$ such that $C', A' \vdash_n e_1' : \sigma' \,\&\, b'$ occurs at $E$ in the inference tree of $C, A \vdash_n E[e_1'] : \sigma \,\&\, b$. By Lemma 3.17 we infer that $A' = A$; this enables us to use Lemma 3.22 from which we get

$$C', A' \vdash_n e_2' : \sigma' \,\&\, b'$$

and by Fact 3.16 we get the desired judgement

$$C, A \vdash_n E[e_2'] : \sigma \,\&\, b.$$

This completes the proof. □

## 3.5 Erroneous Programs cannot be Typed

The purpose of types is to detect certain kinds of errors at analysis time rather than at execution time. To this end one usually (cf. the methodical considerations in [33]) wants a result that guarantees that "error configurations are not typeable"; here we presuppose some consistent constraint set $C$ and some standard channel environment $A$, and assume $A$ is solvable from $C$ so by Lemma 2.24 we need consider only normalised inferences. (The reason for demanding consistency is that otherwise too many expressions may be assigned a type; if e.g. $C$ contains a constraint $\texttt{int} \subseteq \texttt{bool}$ then $C, A \vdash_n \texttt{if } 7 \texttt{ then } 8 \texttt{ else } 9 : \texttt{int} \,\&\, \varepsilon$.)

By Proposition 3.9 and the discussion after it (together with Fact 3.14 and Lemma 3.17), it suffices to consider each of the error configurations listed below, and show that it is not typeable.

<u>$ch\ w$ with $ch$ a channel identifier:</u> since $A(ch)$ is of the form $t\ \texttt{chan}\ \rho$, in order for $ch\ w$ to be typeable it must be the case that $C \vdash t\ \texttt{chan}\ \rho \subseteq t_1 \ \rightarrow^{\beta}\ t_2$ for some $t_1$,$t_2$; this conflicts with $C$ being consistent.

<u>$\texttt{if}\ w\ \texttt{then}\ e_1\ \texttt{else}\ e_2$ with $w \notin \{\texttt{true}, \texttt{false}\}$:</u> for this to be typeable it must hold that

$w$ can be assigned the type $\texttt{bool}$.

As $C$ is consistent we infer that $w$ cannot be a channel identifier (as $A$ is a channel environment) and that $w$ cannot be a function abstraction. Hence $w$ is of the form $C^n\!<\!w_1, \cdots, w_n\!>$; and an examination of Figure 2.4 (using Lemma 3.18) will reveal that this can be given type $\texttt{bool}$ only when $n = 0$ and $C^n \in \{\texttt{true}, \texttt{false}\}$.

<u>$(C^n\!<\!w_1, \cdots, w_n\!>)\ w$:</u> for this to be typeable it must hold that

$C^n\!<\!w_1, \cdots, w_n\!>$ can be assigned a type of the form $t_1 \ \rightarrow^{\beta}\ t_2$

and (using Lemma 3.18 and Fact 2.16) it is easy to see that this is impossible.

<u>$F_s\!<\!w\!>$ with $\delta(F_s\!<\!w\!>)$ undefined:</u> consider e.g. the expression $\texttt{fst}\!<\!w\!>$. For this to be typeable there must (Lemma 3.18) exist $t_1$ and $t_2$ such that

$w$ can be assigned the type $t_1 \times t_2$.

As $C$ is consistent we infer that $w$ cannot be a channel identifier (as $A$ is a channel environment) and that $w$ cannot be a function abstraction. Hence $w$ is of the form $C^n \!<\! w_1, \cdots, w_n \!>$; and an examination of Figure 2.4 (using Lemma 3.18) will reveal that it must be the case that $C^n = \texttt{pair}$. Thus $w$ is of the form $\texttt{pair}\!<\! w_1, w_2 \!>$, but then $\delta(\texttt{fst}\!<\! w \!>)$ is not undefined.

## 3.6 Concurrent Soundness

First a crucial result which generalises Fact 3.16 in two ways:

- The "new" expression $e_n$ may be typed using an environment which is an *extension* of the environment in which the old expression $e$ was typed. Such an extension is a potential danger to semantic soundness, cf. the considerations in [30, section 5] where it was concluded that store operations in Standard ML are harmless unless they actually expand the store; in order to construct an inference tree with the new environment we must demand that the new environment variables are "present" in the behaviour.
- The "new" expression $e_n$ may have a behaviour which is a "suffix" of the behaviour of the old expression $e$, the corresponding "prefix" represents the "action" of going from $e$ to $e_n$.

**Lemma 3.24** Suppose the judgement $jdg' = (C', A \vdash e : \sigma' \,\&\, b')$ occurs at $E$ in the normalised inference $jdg = (C, A \vdash_n E[e] : \sigma \,\&\, b)$ where $C$ (and by Lemma 3.17 then also $C'$) is well-formed and consistent.

Let $b_n$ be a behaviour and let $A_n$ be of the form $A[x_1 : \sigma_1][\cdots : \cdots][x_m : \sigma_m]$ with $m \geq 0$, such that $x_1 \cdots x_m$ do not occur in $E[e]$ and such that $FV(\sigma_1) \cup \cdots \cup FV(\sigma_m) \subseteq FV(b_n)$.

Let $e_n$ be an expression and $b'_r$ a behaviour such that

$$C', A_n \vdash_n e_n : \sigma' \,\&\, b'_r \text{ and}$$

$$C' \vdash b_n; b'_r \subseteq b'.$$

Then there exists $b_r$ such that

$$C, A_n \vdash_n E[e_n] : \sigma \,\&\, b_r \text{ and}$$
$$C \vdash b_n; b_r \subseteq b.$$

Moreover, there exists $S$ with $Dom(S) \cap FV(A, b_n) = \emptyset$ such that $C \vdash S\, C'$.

**Proof** The full proof is given in Appendix B; here we only consider the crucial case where $jdg$ follows from $jdg'$ by an application of (gen). The situation is

$$\frac{jdg' = C \cup C_0, A \vdash e : t_0 \,\&\, b}{jdg = C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}$$

where $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is well-formed and where $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$ and where there exists $S_0$ with $Dom(S_0) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ such that $C \vdash S_0\, C_0$. Our assumptions are

$$C \cup C_0, A_n \vdash_n e_n : t_0 \,\&\, b_r' \quad (1)$$
$$C \cup C_0 \vdash b_n; b_r' \subseteq b \quad (2)$$

and we must show that there exists $b_r$ and $S$ such that the following holds:

$$C, A_n \vdash_n e_n : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b_r \quad (3)$$
$$C \vdash b_n; b_r \subseteq b \quad (4)$$
$$Dom(S) \cap FV(A, b_n) = \emptyset \text{ and } C \vdash S\,(C \cup C_0).$$

We choose $b_r = b_r'$ and $S = S_0$ and then it will suffice to prove

$$\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(b_n, b_r') = \emptyset \quad (5)$$

for then (2) and Lemma 2.18 give that $C \cup S\, C_0 \vdash b_n; b_r' \subseteq b$ which (by Lemma 2.19) implies (4); and since $FV(A_n) \setminus FV(A) \subseteq FV(b_n)$ holds by assumption we will be able to use (gen) to arrive at (3) from (1).

So we are left with the task of proving (5). By the assumption $FV(b) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$ this can be done by showing

$$\forall \gamma' \in FV(b_n, b_r') \exists \gamma \in FV(b) : C \cup C_0 \vdash \gamma' \leftarrow^* \gamma \quad (6)$$
$$\forall \gamma' \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\} : C \cup C_0 \vdash \gamma' \leftarrow^* \gamma \text{ implies } \gamma \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}. \quad (7)$$

(6) follows from (2) by Lemma 2.29, since $C \cup C_0$ is well-formed and consistent. (7) follows from Lemma 2.33. □

Next some auxiliary results concerning the three kinds of concurrent transitions:

**Lemma 3.25** Let $C$ be well-formed and consistent, let $A$ be standard, and suppose that

$$C, A \vdash_n E[\texttt{channel}^l < \texttt{()} >] \,:\, \sigma \,\&\, b.$$

Let $ch$ be a channel identifier that does not occur in $E[\texttt{channel}^l < \texttt{()} >]$; then there exists $t_n$, $\rho_n$ and $b_r$ such that

$$C \vdash t_n \text{ CHAN } \rho_n; b_r \subseteq b \text{ and } C \vdash \{l\} \subseteq \rho_n \text{ and}$$
$$C, A[ch : t_n \texttt{ chan } \rho_n] \vdash_n E[ch] \,:\, \sigma \,\&\, b_r.$$

**Proof** The normalised inference tree contains a judgement of the form

$$C', A \vdash_n \texttt{channel}^l < \texttt{()} > \,:\, t' \,\&\, b'$$

where $C'$ is well-formed and consistent (Lemma 3.17). Since $A$ is standard $A(\texttt{channel}^l)$ is given by

$$\forall(\alpha\beta\rho : \{\alpha \text{ CHAN } \rho \subseteq \beta,\, \{l\} \subseteq \rho\}).\ \texttt{unit} \ \to^\beta\ (\alpha \texttt{ chan } \rho)$$

so using Lemma 3.19 (and subsequently Lemma 3.21 on the typing of `()`) we infer that there exists $S$ such that

$$C' \vdash S\,\alpha \text{ CHAN } S\,\rho \subseteq S\,\beta \text{ and } C' \vdash \{l\} \subseteq S\,\rho \tag{8}$$
$$C' \vdash S\,\alpha \texttt{ chan } S\,\rho \subseteq t' \text{ and } C' \vdash S\,\beta \subseteq b'.$$

Now define $t_n = S\,\alpha$ and $\rho_n = S\,\rho$ and $b_n = t_n \text{ CHAN } \rho_n$, then

$$C', A[ch : t_n \texttt{ chan } \rho_n] \vdash_n ch \,:\, t' \,\&\, \varepsilon \text{ and } C' \vdash b_n; \varepsilon \subseteq b'$$

so as $FV(t_n \texttt{ chan } \rho_n) \subseteq FV(b_n)$ Lemma 3.24 gives us $b_r$ such that

$$C, A[ch : t_n \; \texttt{chan} \; \rho_n] \vdash_n E[ch] : \sigma \,\&\, b_r \text{ and } C \vdash t_n \; \text{CHAN} \; \rho_n ; b_r \subseteq b$$

and additionally $S'$ with $Dom(S') \cap FV(b_n) = \emptyset$ such that $C \vdash S' \, C'$; using Lemmas 2.18 and 2.19 on (8) we therefore get $C \vdash \{l\} \subseteq \rho_n$ and this completes the proof. □

**Lemma 3.26** Let $C$ be well-formed and consistent, let $A$ be standard, and suppose that

$$C, A \vdash_n E[\texttt{spawn}<w>] : \sigma \,\&\, b.$$

Then there exists $b_r$, $t''$, $b''$ such that

**(a)** $C, A \vdash_n E[\texttt{()}] : \sigma \,\&\, b_r$;

**(b)** $C, A \vdash_n w \; \texttt{()} : t'' \,\&\, b''$;

**(c)** $C \vdash (SPAWN \, b''); b_r \subseteq b$.

**Proof** The normalised inference tree contains a judgement of the form

$$C', A \vdash_n \texttt{spawn}<w> : t' \,\&\, b'$$

where $C'$ is well-formed and consistent (Lemma 3.17).

Since $A(\texttt{spawn}) = \forall(\alpha\beta\beta_0 : \{SPAWN \, \beta_0 \subseteq \beta\}). \; (\texttt{unit} \rightarrow^{\beta_0} \alpha) \rightarrow^{\beta} \texttt{unit}$, we from Lemma 3.19 get $t_1$, $b_1$ and $S$ such that

$$C' \vdash SPAWN \, (S \, \beta_0) \subseteq S \, \beta \tag{9}$$

$$C' \vdash \texttt{unit} \subseteq t' \text{ and } C' \vdash b_1 ; S \, \beta \subseteq b' \tag{10}$$

$$C', A \vdash_n w : t_1 \,\&\, b_1 \text{ and } C' \vdash t_1 \subseteq \texttt{unit} \rightarrow^{S \, \beta_0} S \, \alpha \tag{11}$$

and by Lemma 3.21 we infer that

$$C' \vdash \varepsilon \subseteq b_1 \text{ and } C', A \vdash_n w : t_1 \,\&\, \varepsilon. \tag{12}$$

From (10) we therefore get

$$C', A \vdash_n \texttt{()} : t' \,\&\, \varepsilon \text{ and } C' \vdash S\,\beta; \varepsilon \equiv \varepsilon; S\,\beta \subseteq b'$$

and Lemma 3.24 (with $m = 0$ and $b_n = S\,\beta$) then gives us a $b_r$ such that

$$C, A \vdash_n E[\texttt{()}] : \sigma \,\&\, b_r \text{ and } C \vdash S\,\beta; b_r \subseteq b \tag{13}$$

which yields the claim (a), and in addition an $S'$ such that

$$\mathit{Dom}(S') \cap \mathit{FV}(A, S\,\beta) = \emptyset \text{ and } C \vdash S'\,C'. \tag{14}$$

For the remaining claims, we from (11) and (12) infer that

$$C', A \vdash_n w \texttt{ ()} : S\,\alpha \,\&\, S\,\beta_0$$

so using (14) we (by Lemma 2.18 and Lemma 2.19) arrive at

$$C, A \vdash_n w \texttt{ ()} : t'' \,\&\, b''$$

for $t'' = S'\,S\,\alpha$ and $b'' = S'\,S\,\beta_0$, thus yielding the claim (b).

In order to show (c) it by (13) is sufficient to show that $C \vdash \mathit{SPAWN}\, b'' \subseteq S\,\beta$. But this follows from (9) using (14) (by Lemma 2.18 and 2.19). □

**Lemma 3.27** Let $C$ be well-formed and consistent, let $A$ be standard and a channel environment, and suppose that

$$C, A \vdash_n E_1[\texttt{sync}<\texttt{transmit}<\texttt{pair}<ch, w>>>] : \sigma_1 \,\&\, b_1 \tag{15}$$

and that

$$C, A \vdash_n E_2[\texttt{sync}<\texttt{receive}<ch>>] : \sigma_2 \,\&\, b_2. \tag{16}$$

Let $A(ch) = t \texttt{ chan } \rho_0$, then there exists $t_s$, $b_s$, $\rho_s$ and $t_r$, $b_r$, $\rho_r$ such that

**(a)** $C, A \vdash_n E_1[w] : \sigma_1 \,\&\, b_s$ and

$C \vdash \rho_s\,!\,t_s; b_s \subseteq b_1$ and $C \vdash t_s \subseteq t$ and $C \vdash \rho_0 \subseteq \rho_s$;

**(b)** $C, A \vdash_n w : t \,\&\, \varepsilon$;

**(c)** $C, A \vdash_n E_2[w] : \sigma_2 \,\&\, b_r$ and

$C \vdash \rho_r \,?\, t_r; b_r \subseteq b_2$ and $C \vdash t \subseteq t_r$ and $C \vdash \rho_0 \subseteq \rho_r$.

**Proof** The tree (15) will contain a judgement of the form

$$C_1, A \vdash_n \texttt{sync}\!<\!\texttt{transmit}\!<\!\texttt{pair}\!<\!ch, w\!>\!>\!> : t_1 \,\&\, b_1' \quad (17)$$

where $C_1$ is well-formed and consistent (Lemma 3.17). Since

$$A(\texttt{sync}) = \forall(\alpha\beta : \emptyset).\ (\alpha\ \texttt{event}\ \beta) \ \rightarrow^{\beta}\ \alpha$$

Lemma 3.19 together with Lemma 3.21 tells us that there exists $t_3$ and $S_3$ such that

$$C_1, A \vdash_n \texttt{transmit}\!<\!\texttt{pair}\!<\!ch, w\!>\!> : t_3 \,\&\, \varepsilon;$$

$$C_1 \vdash S_3\,\beta \subseteq b_1';$$

$$C_1 \vdash S_3\,\alpha \subseteq t_1;$$

$$C_1 \vdash t_3 \subseteq (S_3\,\alpha)\ \texttt{event}\ (S_3\,\beta).$$

As $A(\texttt{transmit}) = \forall(\alpha\beta\rho : \{\rho\,!\,\alpha \subseteq \beta\}).\ (\alpha\ \texttt{chan}\ \rho) \times \alpha \ \rightarrow\ (\alpha\ \texttt{event}\ \beta)$, Lemma 3.18 (together with Lemma 3.21) tells us that there exists $t_4$ and $S_4$ such that

$$C_1, A \vdash_n \texttt{pair}\!<\!ch, w\!> : t_4 \,\&\, \varepsilon;$$

$$C_1 \vdash (S_4\,\rho)\,!\,(S_4\,\alpha) \subseteq S_4\,\beta;$$

$$C_1 \vdash (S_4\,\alpha)\ \texttt{event}\ (S_4\,\beta) \subseteq t_3;$$

$$C_1 \vdash t_4 \subseteq (S_4\,\alpha)\ \texttt{chan}\ (S_4\,\rho) \times (S_4\,\alpha).$$

Since $A(\texttt{pair}) = \forall(\alpha_1\alpha_2 : \emptyset).\ \alpha_1 \ \rightarrow\ \alpha_2 \ \rightarrow\ \alpha_1 \times \alpha_2$, Lemma 3.18 (together with Lemma 3.21) tells us that there exists $t_5$, $t_6$, and $S_5$ such that

$$C_1, A \vdash_n ch : t_5 \,\&\, \varepsilon; \tag{18}$$

$$C_1, A \vdash_n w : t_6 \,\&\, \varepsilon; \tag{19}$$

$$C_1 \vdash S_5\,\alpha_1 \times S_5\,\alpha_2 \subseteq t_4;$$

$$C_1 \vdash t_5 \subseteq S_5\,\alpha_1 \text{ and } C_1 \vdash t_6 \subseteq S_5\,\alpha_2.$$

Since $A(ch) = t\ \texttt{chan}\ \rho_0$ we infer from (18) that

$$C_1 \vdash t\ \texttt{chan}\ \rho_0 \subseteq t_5.$$

We now repeatedly apply the rules labelled (bw) from Figs. 2.6–2.8: from

$$C_1 \vdash (S_4\,\alpha)\ \texttt{event}\ (S_4\,\beta) \subseteq t_3 \subseteq (S_3\,\alpha)\ \texttt{event}\ (S_3\,\beta)$$

$$C_1 \vdash t_5 \times t_6 \subseteq S_5\,\alpha_1 \times S_5\,\alpha_2 \subseteq t_4 \subseteq (S_4\,\alpha)\ \texttt{chan}\ (S_4\,\rho) \times (S_4\,\alpha)$$

we deduce that

$$C_1 \vdash S_4\,\alpha \subseteq S_3\,\alpha \subseteq t_1$$

$$C_1 \vdash t\ \texttt{chan}\ \rho_0 \subseteq t_5 \subseteq (S_4\,\alpha)\ \texttt{chan}\ (S_4\,\rho) \tag{20}$$

$$C_1 \vdash t_6 \subseteq S_4\,\alpha$$

$$C_1 \vdash S_4\,\beta \subseteq S_3\,\beta.$$

From (19) we therefore get

$$C_1, A \vdash_n w : t_1 \,\&\, \varepsilon$$

so by Lemma 3.24 applied to (15) and (17) we find $b_s$ and $S_1$ such that

$$C, A \vdash_n E_1[w] : \sigma_1 \,\&\, b_s \text{ and } C \vdash b_1'; b_s \subseteq b_1; \tag{21}$$

$$Dom(S_1) \cap FV(A, b_1') = \emptyset \text{ and } C \vdash S_1\,C_1. \tag{22}$$

Let $t_s = S_1\,S_4\,\alpha$ and $\rho_s = S_1\,S_4\,\rho$. By exploiting the *contravariance* of $\cdots$ `chan` (cf. the remarks concerning Figure 2.6), we from (20) get

$$C_1 \vdash t_6 \subseteq S_4\,\alpha \subseteq t$$

and from (19) therefore

$$C_1, A \vdash_n w : t \,\&\, \varepsilon$$

and in addition we have (using (bw) on (20))

$$C_1 \vdash (S_4\,\rho)\,!\,(S_4\,\alpha) \subseteq S_4\,\beta \subseteq S_3\,\beta \subseteq b'_1 \text{ and } C_1 \vdash \rho_0 \subseteq S_4\,\rho.$$

Using (22) we from the preceding lines get (as $FV(t, \rho_0) \subseteq FV(A)$)

$$C \vdash t_s \subseteq t \text{ and } C, A \vdash_n w : t \,\&\, \varepsilon \text{ and}$$

$$C \vdash \rho_s\,!\,t_s \subseteq b'_1 \text{ and } C \vdash \rho_0 \subseteq \rho_s.$$

Together with (21) this yields the claims (a) and (b).

Our remaining task is to show claim (c), where we first notice that the tree (16) will contain a judgement of the form

$$C_2, A \vdash_n \texttt{sync}\!<\!\texttt{receive}\!<\!ch\!>\!> : t_2 \,\&\, b'_2 \tag{23}$$

where $C_2$ is well-formed and consistent (Lemma 3.17). Since

$$A(\texttt{sync}) = \forall(\alpha\beta : \emptyset).\ (\alpha \;\texttt{event}\; \beta) \;\rightarrow^{\beta}\; \alpha$$

Lemma 3.19 (together with Lemma 3.21) tells us that there exists $t_7$ and $S_7$ such that

$$C_2, A \vdash_n \texttt{receive}\!<\!ch\!> : t_7 \,\&\, \varepsilon;$$

$$C_2 \vdash S_7\,\beta \subseteq b'_2;$$

$$C_2 \vdash S_7\,\alpha \subseteq t_2;$$

$$C_2 \vdash t_7 \subseteq (S_7\,\alpha) \;\texttt{event}\; (S_7\,\beta).$$

Since $A(\texttt{receive}) = \forall(\alpha\beta\rho : \{\rho\,?\,\alpha \subseteq \beta\}).\ (\alpha \;\texttt{chan}\; \rho) \rightarrow (\alpha \;\texttt{event}\; \beta)$, Lemma 3.18 tells us that there exists $t_8$ and $S_8$ such that

$$C_2, A \vdash_n ch : t_8 \,\&\, \varepsilon; \tag{24}$$

$$C_2 \vdash (S_8\,\rho)\,?\,(S_8\,\alpha) \subseteq S_8\,\beta;$$

$$C_2 \vdash (S_8\,\alpha)\;\texttt{event}\;(S_8\,\beta) \subseteq t_7;$$

$$C_2 \vdash t_8 \subseteq (S_8\,\alpha)\;\texttt{chan}\;(S_8\,\rho).$$

Since $A(ch) = t\;\texttt{chan}\;\rho_0$ we infer from (24) that

$$C_2 \vdash t\;\texttt{chan}\;\rho_0 \subseteq t_8.$$

We now repeatedly apply the rules labelled (bw) from Figs. 2.6–2.8: from

$$C_2 \vdash (S_8\,\alpha)\;\texttt{event}\;(S_8\,\beta) \subseteq t_7 \subseteq (S_7\,\alpha)\;\texttt{event}\;(S_7\,\beta)$$

$$C_2 \vdash t\;\texttt{chan}\;\rho_0 \subseteq t_8 \subseteq (S_8\,\alpha)\;\texttt{chan}\;(S_8\,\rho)$$

we get, by exploiting the *covariance* of $\cdots$ `chan` (cf. the remarks concerning Figure 2.6),

$$C_2 \vdash t \subseteq S_8\,\alpha \subseteq S_7\,\alpha \subseteq t_2 \tag{25}$$

$$C_2 \vdash (S_8\,\rho)\,?\,(S_8\,\alpha) \subseteq S_8\,\beta \subseteq S_7\,\beta \subseteq b_2' \text{ and } C_2 \vdash \rho_0 \subseteq S_8\,\rho. \tag{26}$$

Clearly $C \subseteq C_2$ so by Lemma 2.19 we can deduce from claim (b) that

$$C_2, A \vdash_n w : t \,\&\, \varepsilon$$

so by applying (sub) we arrive at

$$C_2, A \vdash_n w : t_2 \,\&\, \varepsilon.$$

By applying Lemma 3.24 on (16) and (23) we find $b_r$ and $S_2$ such that

$$C, A \vdash_n E_2[w] : \sigma_2 \,\&\, b_r \text{ and } C \vdash b_2'; b_r \subseteq b_2; \tag{27}$$

$$Dom(S_2) \cap FV(A, b_2') = \emptyset \text{ and } C \vdash S_2\,C_2. \tag{28}$$

Let $t_r = S_2\,S_8\,\alpha$ and $\rho_r = S_2\,S_8\,\rho$. By (28) we from (25) and (26) get (as $FV(t, \rho_0) \subseteq FV(A)$)

$$C \vdash t \subseteq t_r \text{ and } C \vdash \rho_r\,?\,t_r \subseteq b_2' \text{ and } C \vdash \rho_0 \subseteq \rho_r$$

which together with (27) yields the claim (c).

This completes the proof. □

We are now able to formulate that our system is semantically sound, in the sense that "well-typed programs communicate according to their behaviour". We write $C, A \vdash PP : PT \,\&\, PB$, where $PT$ (respectively $PB$) is a mapping from process identifiers into types (respectively behaviours), if the domains of $PP$, $PT$ and $PB$ are equal and if for all $p \in \mathit{Dom}(PP)$ we have $C, A \vdash PP(p) : PT(p) \,\&\, PB(p)$.

**Theorem 3.28** *Semantic (concurrent) soundness*

Let $C$ be well-formed and consistent, let $A$ be a standard channel environment, and suppose

$$C, A \vdash_n PP : PT \,\&\, PB.$$

If $PP \overset{sa}{\Longleftrightarrow} PP'$ then there exists $PT'$, $PB'$ and a standard channel environment $A'$ such that

$$C, A' \vdash_n PP' : PT' \,\&\, PB'$$

and such that if $ch$ occurs in $PP$ then $A'(ch) = A(ch)$ and such that if $p$ is in the domain of $PP$ then *(i)* $PT'(p) = PT(p)$ and *(ii)* if $p$ is not mentioned in $sa$ then $PB'(p) = PB(p)$.

Furthermore we have the following property:

- If $sa = p_0 \;\texttt{chan}^l\; ch_0$ then there exists $t_0$ and $\rho_0$ such that $A'(ch_0) = t_0 \;\texttt{chan}\; \rho_0$ and such that

  $$C \vdash t_0 \;\text{CHAN}\; \rho_0; PB'(p_0) \subseteq PB(p_0) \text{ and } C \vdash \{l\} \subseteq \rho_0.$$

- If $sa = p_0 \;\texttt{spawn}\; p'$ then

  $$C \vdash (\mathit{SPAWN}\, PB'(p')); PB'(p_0) \subseteq PB(p_0).$$

- If $sa = p_1, p_2 \;\texttt{comm}\; ch_0$ then, with $A(ch_0) = t \;\texttt{chan}\; \rho$, there exists

$t_s$ and $t_r$ with $C \vdash t_s \subseteq t \subseteq t_r$ and

$\rho_s$ and $\rho_r$ with $C \vdash \rho \subseteq \rho_s$ and $C \vdash \rho \subseteq \rho_r$

such that

$$C \vdash (\rho_s \,!\, t_s); PB'(p_1) \subseteq PB(p_1)$$

$$C \vdash (\rho_r \,?\, t_r); PB'(p_2) \subseteq PB(p_2).$$

**Proof** We perform case analysis on the semantic action $sa$:

$sa = \texttt{seq}$: It follows from Theorem 3.23 that we can use $PT' = PT$, $PB' = PB$ and $A' = A$.

$sa = p_0\ \texttt{chan}^l\ ch_0$: It follows from Lemma 3.25 that there exists $t_0$,$\rho_0$ and $b_r$ such that the claim follows with $PT' = PT$, $PB' = PB[p_0 : b_r]$ and $A' = A[ch_0 : t_0\ \texttt{chan}\ \rho_0]$. (For $p$ in the domain of $PP$ with $p \neq p_0$ we must show that $C, A \vdash_n PP(p) : PT(p) \,\&\, PB(p)$ implies $C, A' \vdash_n PP(p) : PT(p) \,\&\, PB(p)$, but this follows from Fact 2.21.)

$sa = p_0\ \texttt{spawn}\ p'$: It follows from Lemma 3.26 that there exists $t''$, $b''$ and $b_r$ such that we can use $PT' = PT[p' : t'']$, $PB' = PB[p_0 : b_r][p' : b'']$ and $A' = A$.

$sa = p_1, p_2\ \texttt{comm}\ ch_0$: It follows from Lemma 3.27 that there exists $b_s$ and $b_r$ such that we can use $PT' = PT$, $PB' = PB[p_1 : b_s][p_2 : b_r]$ and $A' = A$. □

**Remark** Theorem 3.28 makes it explicit that the type of a channel does not change after it has been allocated. This should be compared with the subject reduction result in [33, Lemma 5.2], the formulation of which allows one the possibility of assigning different types to the same location at various stages (although apparently it is always possible to choose the same type and still get subject reduction). □

# Chapter 4

# The Inference Algorithm

In designing an inference algorithm $\mathcal{W}$ for the type inference system we are motivated by the overall approach of [26, 6]. One ingredient (called $\mathcal{W}'$) of this will be to perform a syntax-directed traversal of the expression in order to determine its type and behaviour; this will involve constructing a constraint set for expressing the required relationship between the type and behaviour and region variables. The second ingredient (called $\mathcal{F}$) will be to perform a decomposition of the constraint set into one that is *atomic* (as to be explained below). The third ingredient (called $\mathcal{R}$) amounts to (significantly) reducing the constraint set; this is optional and a somewhat open ended endeavour.

The algorithm also employs the notion of well-formedness for constraint sets (introduced in Definition 2.10) and for types and type schemes (introduced in Definition 2.12). Recall (Fact 2.11 and Fact 2.13) that these notions are closed under substitution; this will *not* be the case for the notion of atomicity.

## Atomicity

As in [12, 6, 26] we shall want the type constraints to *match* and shall decompose them into *atomic* constraints; in our setting these will not contain base types as we have no ordering among those.

**Definition 4.1** A constraint set $C$ is *atomic* if *(i)* $C$ is well-formed, and *(ii)* all type constraints in $C$ are of the form $\alpha_1 \subseteq \alpha_2$. □

Atomicity is a rather strong notion:

**Fact 4.2** Let $C$ be atomic. Then $C$ is also well-formed and consistent; and it holds that $(\overline{C})^b = C^b$ and $(\overline{C})^r = C^r$; so if $C \vdash \gamma \leftarrow \beta$ then there exists $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in C$.

**Proof** To prove consistency one may employ the notion of *matching* and the claim will be a corollary[1] of Fact 5.11. □

Atomicity of type constraints is responsible for transforming constraints like $(\alpha \subseteq \texttt{int})$ and $(t_1 \times t_2 \subseteq \alpha)$ by forcing $\alpha$ to be replaced by a type expression that "matches" the opposite side of the constraint, and for disallowing constraints like $(t_1 \times t_2 \subseteq t'_1 \rightarrow^\beta t'_2)$; a phenomenon that can be found in [12, 6, 26] as well. This feature is responsible for making the algorithm a "conservative extension" of the way algorithm $\mathcal{W}$ for Standard ML would operate if effects were not taken into account: in particular our algorithm will fail, rather than produce an unsolvable constraint set, if the underlying type constraints of the effect-free system cannot be solved. (We shall make this point more precise in Section 4.6.)

## 4.1 Algorithm $\mathcal{W}$

Our key algorithm $\mathcal{W}$ is described by

$$\mathcal{W}(A, e) = (S, t, b, C)$$

where the intuition is that $C, S\,A \vdash e : t\,\&\,b$ is the "best correct" typing of $e$ relative to an assumption list derived from $A$. We shall enforce throughout (by using $\mathcal{F}$) that $C$ is atomic provided that $A$ is well-formed. Algorithm $\mathcal{W}$ is defined by the clause

$$\begin{aligned}\mathcal{W}(A, e) = {} & \text{let } (S_1, t_1, b_1, C_1) = \mathcal{W}'(A, e) \\ & \text{let } (S_2, C_2) = \mathcal{F}(C_1) \\ & \text{let } (C_3, t_3, b_3) = \mathcal{R}(C_2, S_2\,t_1, S_2\,b_1, S_2\,S_1\,A) \\ & \text{in } (S_2\,S_1, t_3, b_3, C_3)\end{aligned}$$

[1] There is no circularity going on, but to state Fact 5.11 already now requires us to set up some amount of machinery and we will rather postpone this.

where the definitions of $\mathcal{W}'$ and $\mathcal{W}$ are mutually recursive; algorithm $\mathcal{W}'$ is responsible for the syntax-directed traversal of the argument $e \in EExp$. In general, $\mathcal{W}'$ will fail to produce an atomic constraint set $C$, even when the assumption list $A$ is well-formed; it will be the case, however, that $C$ is well-formed. This then motivates the need for a transformation $\mathcal{F}$ (Sect. 4.3) that maps a well-formed constraint set into an atomic constraint set; since this involves splitting variables we shall need to produce a substitution as well. The final transformation $\mathcal{R}$ merely attempts to get a smaller constraint set by removing variables that are not strictly needed. Its operation is not essential for the soundness or completeness of our algorithm and thus one might define it by $\mathcal{R}(C, t, b, A) = (C, t, b)$; in Sect. 4.4 we shall consider a more powerful version of $\mathcal{R}$.

**Example 4.3** To make the intentions a bit clearer suppose that $\mathcal{W}'(A, e) = (S_1, t_1, b_1, C_1)$ so that $C_1, S_1\, A \vdash e : t_1 \,\&\, b_1$ is the "best correct" typing of $e$. If

$$C_1 = \{\alpha_1 \times \alpha_2 \subseteq \alpha_3,\ \alpha_4 \subseteq \texttt{int},\ \alpha_5 \text{ CHAN } \rho; \varepsilon \subseteq \beta\}$$

then $(S_2, C_2) = \mathcal{F}(C_1)$ should give

$$C_2 = \{\alpha_1 \subseteq \alpha_{31},\ \alpha_2 \subseteq \alpha_{32},\ \alpha_5 \text{ CHAN } \rho; \varepsilon \subseteq \beta\}$$
$$S_2 = [\alpha_3 \mapsto \alpha_{31} \times \alpha_{32},\ \alpha_4 \mapsto \texttt{int}]$$

We expand $\alpha_3$ to $\alpha_{31} \times \alpha_{32}$ so the resulting constraint $\alpha_1 \times \alpha_2 \subseteq \alpha_{31} \times \alpha_{32}$ can be "decomposed" into the atomic constraints $\alpha_1 \subseteq \alpha_{31}$ and $\alpha_2 \subseteq \alpha_{32}$. Furthermore we have expanded $\alpha_4$ to `int` as it follows from Figure 2.6 that $\emptyset \vdash t \subseteq \texttt{int}$ necessitates that $t$ equals `int`. Clearly the intention is that also $C_2, S_2\, S_1\, A \vdash e : S_2\, t_1 \,\&\, S_2\, b_1$ is the "best correct" typing of $e$ and additionally the constraint set is atomic (unlike what is the case for $C_1$). □

## 4.2 Algorithm $\mathcal{W}'$

Algorithm $\mathcal{W}'$ is defined by the clauses in Figure 4.1 and is to be defined simultaneously with $\mathcal{W}$ since it calls $\mathcal{W}$ in a number of places. Actually it

$\mathcal{W}'(A, c) = \text{if } c \in \mathit{Dom}(A) \text{ then } \mathit{INST}(A(c)) \text{ else } \mathit{fail}_{const}$

$\mathcal{W}'(A, x) = \text{if } x \in \mathit{Dom}(A) \text{ then } \mathit{INST}(A(x)) \text{ else } \mathit{fail}_{ident}$

$$
\begin{aligned}
&\mathcal{W}'(A, \texttt{fn } x \Rightarrow e_0) = \\
&\qquad \text{let } \alpha \text{ be fresh} \\
&\qquad \text{let } (S_0, t_0, b_0, C_0) = \mathcal{W}(A[x : \alpha], e_0) \\
&\qquad \text{let } \beta \text{ be fresh} \\
&\qquad \text{in } (S_0, S_0\, \alpha \;\to^{\beta}\; t_0, \varepsilon, C_0 \cup \{b_0 \subseteq \beta\})
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{W}'(A, e_1\ e_2) = \\
&\qquad \text{let } (S_1, t_1, b_1, C_1) = \mathcal{W}(A, e_1) \\
&\qquad \text{let } (S_2, t_2, b_2, C_2) = \mathcal{W}(S_1\, A, e_2) \\
&\qquad \text{let } \alpha, \beta \text{ be fresh} \\
&\qquad \text{in } (S_2\, S_1, \alpha, (S_2\, b_1; b_2; \beta), S_2\, C_1 \cup C_2 \cup \{S_2\, t_1 \subseteq t_2 \;\to^{\beta}\; \alpha\})
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{W}'(A, e_0 \,@^{s}_{n} < e_1, \cdots, e_n >) = \\
&\qquad \cdots \text{let } (S_i, t_i, b_i, C_i) = \mathcal{W}(S_{i-1} \cdots S_1\, S_0\, A, e_i) \cdots \text{let } \alpha \text{ be fresh} \\
&\qquad \text{in } (S_n \cdots S_1\, S_0, \alpha, (S_n \cdots S_1\, b_0; S_n \cdots S_2\, b_1; \cdots; b_n), \\
&\qquad\qquad \cdots \cup S_n \cdots S_{i+1}\, C_i \cup \cdots \cup \{S_n \cdots S_1\, t_0 \subseteq S_n \cdots S_2\, t_1 \;\to\; \cdots t_n \;\to\; \alpha\})
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{W}'(A, \texttt{let } x = e_1 \texttt{ in } e_2) = \\
&\qquad \text{let } (S_1, t_1, b_1, C_1) = \mathcal{W}(A, e_1) \\
&\qquad \text{let } ts_1 = \mathit{GEN}(S_1\, A, b_1)(C_1, t_1) \\
&\qquad \text{let } (S_2, t_2, b_2, C_2) = \mathcal{W}((S_1\, A)[x : ts_1], e_2) \\
&\qquad \text{in } (S_2\, S_1, t_2, (S_2\, b_1; b_2), S_2\, C_1 \cup C_2)
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{W}'(A, \texttt{rec } f\ x \Rightarrow e_0) = \\
&\qquad \text{let } \alpha_1, \beta, \alpha_2 \text{ be fresh} \\
&\qquad \text{let } (S_0, t_0, b_0, C_0) = \mathcal{W}(A[f : \alpha_1 \;\to^{\beta}\; \alpha_2][x : \alpha_1], e_0) \\
&\qquad \text{in } (S_0, S_0\, (\alpha_1 \;\to^{\beta}\; \alpha_2), \varepsilon, C_0 \cup \{b_0 \subseteq S_0\, \beta,\ t_0 \subseteq S_0\, \alpha_2\})
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{W}'(A, \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2) = \\
&\qquad \text{let } (S_0, t_0, b_0, C_0) = \mathcal{W}(A, e_0) \\
&\qquad \text{let } (S_1, t_1, b_1, C_1) = \mathcal{W}(S_0\, A, e_1) \\
&\qquad \text{let } (S_2, t_2, b_2, C_2) = \mathcal{W}(S_1\, S_0\, A, e_2) \\
&\qquad \text{let } \alpha \text{ be fresh} \\
&\qquad \text{in } (S_2\, S_1\, S_0, \alpha, (S_2\, S_1\, b_0; (S_2\, b_1 + b_2)), \\
&\qquad\qquad S_2\, S_1\, C_0 \cup S_2\, C_1 \cup C_2 \cup \{S_2\, S_1\, t_0 \subseteq \texttt{bool},\ S_2\, t_1 \subseteq \alpha,\ t_2 \subseteq \alpha\})
\end{aligned}
$$

Figure 4.1: Syntax-directed constraint generation

could call itself recursively, rather than calling $\mathcal{W}$, in all but one place[2]: the call to $\mathcal{W}$ immediately prior to the use of *GEN* to generalise the type of the `let`-bound identifier to a type scheme. The algorithm follows the overall approach of [26, 8] except that as in [6] there are no explicit unification steps; these all take place as part of the $\mathcal{F}$ transformation. The main novel ingredient of our approach shows up in the clause for `let` as we shall explain shortly. Concentrating on the overall picture we thus have clauses for constants and identifiers; both make use of the auxiliary function *INST* defined by

$$\begin{array}{lcl} \mathit{INST}(\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t) & = & \text{let } \vec{\alpha'}\vec{\beta'}\vec{\rho'} \text{ be fresh} \\ & & \text{let } R = [\vec{\alpha}\vec{\beta}\vec{\rho} \mapsto \vec{\alpha'}\vec{\beta'}\vec{\rho'}] \\ & & \text{in } (\mathrm{Id}, R\,t, \varepsilon, R\,C) \\ \\ \mathit{INST}(t) & = & (\mathrm{Id}, t, \varepsilon, \emptyset) \end{array}$$

in order to produce a fresh instance of the relevant type or type scheme as determined by the environment $A$. The clause for function abstraction is rather straightforward; note the use of a constraint to record the "meaning" of the fresh behaviour variable. Also the clause for (silent and non-silent) application is rather straightforward; note that instead of a unification step we record the desired connection between the operator and operand types by means of a (non-atomic) constraint. The clauses for recursion and conditional follow the same pattern as the clauses for abstraction and application.

The only novelty in the clause for `let` is the function *GEN* used for generalisation:

$$\begin{array}{lcl} \mathit{GEN}(A, b)(C, t) & = & \text{let } \{\vec{\alpha}\vec{\beta}\vec{\rho}\} = (\mathit{Clos}(FV(t), C)) \setminus (FV(A, b)^{C\downarrow}) \\ & & \text{let } C_0 = C \mid_{\{\vec{\alpha}\vec{\beta}\vec{\rho}\}} \\ & & \text{in } \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t \end{array}$$

where

- $\mathit{Clos}(X, C) = \{\gamma \mid \exists \gamma' \in X : \gamma \sim_C \gamma'\}$ with $\sim_C$ the least equivalence relation satisfying that if $(g_1 \subseteq g_2) \in C$ and $\gamma, \gamma' \in FV(g_1, g_2)$ then $\gamma \sim_C \gamma'$;

[2] This is exactly the place where the algorithm of [26] makes use of constraint simplification in the "*close*" function.

- $C\mid_{\{\vec{\alpha}\vec{\beta}\vec{\rho}\}} = \{(g_1 \subseteq g_2) \in C \mid FV(g_1, g_2) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \neq \emptyset\}$.

The definition of $C_0$ thus establishes the part of the well-formedness condition that requires each constraint to involve at least one bound variable.

The exclusion of the set $FV(A,b)^{C\downarrow}$ (rather than just $FV(A,b)$) is necessary in order to ensure $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{C_0\uparrow} = \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ which is essential for semantic soundness (cf. the discussion in the Introduction); the computation of "Indirect Free Variables" of [32] is very similar to our notion of downwards closure. Finally we have chosen $Clos(FV(t), C)$ as the "universe" in which to perform the set difference; this universe must be large enough that we may still hope for syntactic completeness and all of $FV(t)$, $FV(t)^{C\downarrow}$ (similar to what is in fact taken in [32]) and $FV(t)^{C\uparrow}$ are apparently too small for this (except for the latter they are not even upwards closed).

**Fact 4.4** Let $\sigma = GEN(A,b)(C,t)$. If $C$ is atomic then $\sigma$ is well-formed.

**Proof** Using the terminology from the defining clause for *GEN*, the only non-trivial task is to show that $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}^{C_0\uparrow} \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ (notice that the requirement $FV({C_0}^t) \cap \{\vec{\beta}\} = \emptyset$ would not necessarily hold if we had just assumed $C$ to be well-formed). So assume $C_0 \vdash \gamma \leftarrow \beta$ with $\gamma \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$; we must show that $\beta \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$. From Fact 4.2 we find $b$ with $\gamma \in FV(b)$ such that $(b \subseteq \beta) \in C_0 \subseteq C$, implying $C \vdash \gamma \leftarrow \beta$ and $\gamma \sim_C \beta$. From $\gamma \in Clos(FV(t), C)$ and $\gamma \notin FV(A,b)^{C\downarrow}$ we thus infer $\beta \in Clos(FV(t), C)$ and $\beta \notin FV(A,b)^{C\downarrow}$ so $\beta \in \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ as desired. □

**Remark** Note that $Clos(FV(t), C)$ is a subset of $FV(t, C)$ and that it may well be a proper subset; when this is the case it avoids to generalise over "purely internal" variables that are inconsequential for the overall type. If one were to regard `let` $x = e_1$ `in` $e_2$ as equivalent to $e_2[e_1/x]$ (which is sensible only if $e_1$ has an empty behaviour) this corresponds to forcing all "purely internal" variables in corresponding copies of $e_1$ to be equal. This is helpful for reducing the size of constraint sets and type schemes. □

## 4.3 Algorithm $\mathcal{F}$

We are now going to define the algorithm $\mathcal{F}$ which "forces type constraints to match" by transforming them into atomic constraints; the algorithm closely resembles [6, procedure MATCH].

The algorithm may be described as a non-deterministic rewriting process. It operates over triples of the form $(S, C, \sim)$ where $S$ is a substitution, $C$ is a constraint set, and $\sim$ is an equivalence relation among the finite set of type variables in $C$; we shall write $\mathrm{Eq}_C$ for the identity relation over type variables in $C$. We then define $\mathcal{F}$ by

$$\begin{array}{l} \mathcal{F}(C) = \text{let } (S', C', \sim') \text{ be given by } (\mathrm{Id}, C, \mathrm{Eq}_C) \Leftrightarrow\!\!\!\rightarrow^* (S', C', \sim') \not\Leftrightarrow\!\!\!\rightarrow \\ \qquad\qquad \text{in if all type constraints in } C' \text{ are of the form } \alpha_1 \subseteq \alpha_2 \\ \qquad\qquad\quad \text{then } (S', C') \text{ else } \mathit{fail}_{\mathit{forcing}} \end{array}$$

The rewriting relation is defined by the axioms of Figure 4.3 and will be explained below; it makes use of an auxiliary rewriting relation, defined in Figure 4.2, which operates over constraint sets.

The axioms of Figure 4.2 are rather straightforward, implementing the rules (bw) from Figs. 2.6 and 2.7 and 2.8. (A small notational point: in Figure 4.2 and in Figure 4.3 we write $C \dot{\cup} C'$ for $C \cup C'$ in case $C \cap C' = \emptyset$.)

**Fact 4.5** The rewriting relation $\rightharpoonup$ is confluent and if $C_1 \rightharpoonup C_2$ then $C_2 \vdash C_1$.

**Proof** Confluence follows since each rewriting operates on a single element only, and for each element there is only one possible rewriting. □

We now turn to Figure 4.3. The axiom (dc) decomposes the constraint set but does not modify the substitution nor the equivalence relation among type variables. The axioms (mr) and (ml) both forces left and right hand sides of type constraints to match and produces a new substitution as a result; additionally it may modify the equivalence relation among type variables. The details require the predicate $\mathcal{M}$ (which performs an "occur check"), to be defined shortly. Before presenting the formal definition we consider an example.

$$\left.\begin{array}{ll}(\texttt{unit}) & C \dot{\cup} \{\texttt{unit} \subseteq \texttt{unit}\} \\ (\texttt{bool}) & C \dot{\cup} \{\texttt{bool} \subseteq \texttt{bool}\} \\ (\texttt{int}) & C \dot{\cup} \{\texttt{int} \subseteq \texttt{int}\}\end{array}\right\} \rightharpoonup C$$

$$(\rightarrow) \qquad C \dot{\cup} \{t_1 \rightarrow t_2 \subseteq t_3 \rightarrow t_4\} \rightharpoonup C \cup \{t_3 \subseteq t_1, t_2 \subseteq t_4\}$$

$$(\rightarrow) \qquad C \dot{\cup} \{t_1 \rightarrow^{\beta_1} t_2 \subseteq t_3 \rightarrow^{\beta_2} t_4\}$$
$$\rightharpoonup C \cup \{t_3 \subseteq t_1, \beta_1 \subseteq \beta_2, t_2 \subseteq t_4\}$$

$$(\times) \qquad C \dot{\cup} \{t_1 \times t_2 \subseteq t_3 \times t_4\} \rightharpoonup C \cup \{t_1 \subseteq t_3, t_2 \subseteq t_4\}$$

$$(\texttt{list}) \quad C \dot{\cup} \{t_1\, \texttt{list} \subseteq t_2\, \texttt{list}\} \rightharpoonup C \cup \{t_1 \subseteq t_2\}$$

$$(\texttt{chan}) \quad C \dot{\cup} \{t_1\, \texttt{chan}\, \rho_1 \subseteq t_2\, \texttt{chan}\, \rho_2\} \rightharpoonup C \cup \{t_1 \subseteq t_2, t_2 \subseteq t_1, \rho_1 \subseteq \rho_2\}$$

$$(\texttt{event}) \quad C \dot{\cup} \{t_1\, \texttt{event}\, \beta_1 \subseteq t_2\, \texttt{event}\, \beta_2\} \rightharpoonup C \cup \{t_1 \subseteq t_2, \beta_1 \subseteq \beta_2\}$$

Figure 4.2: Decomposition of constraints

$$(\texttt{dc}) \ \frac{C \rightharpoonup C'}{(S, C, \sim) \Leftrightarrow\!\!\!\rightarrow (S, C', \sim)}$$

$$(\texttt{mr}) \ (S, C \dot{\cup} \{t \subseteq \alpha\}, \sim) \Leftrightarrow\!\!\!\rightarrow (R\,S, R\,C \cup \{R\,t \subseteq R\,\alpha\}, \sim')$$
provided $\mathcal{M}(\alpha, t, \sim, R, \sim')$

$$(\texttt{ml}) \ (S, C \dot{\cup} \{\alpha \subseteq t\}, \sim) \Leftrightarrow\!\!\!\rightarrow (R\,S, R\,C \cup \{R\,\alpha \subseteq R\,t\}, \sim')$$
provided $\mathcal{M}(\alpha, t, \sim, R, \sim')$

Figure 4.3: Rewriting rules for $\mathcal{F}$: forcing well-formedness

**Example 4.6** With $t_1 = (\alpha_{11} \times \alpha_{12})$ `event` $\beta_1$, consider the constraint $t_1 \subseteq \alpha_0$. Forcing the left and right hand sides to match means finding a substitution $R$ such that $R\,t_1$ and $R\,\alpha_0$ have the same shape. A natural way to achieve this is by creating new type variables $\alpha_{21}$ and $\alpha_{22}$ and a new behaviour variable $\beta_2$ and by defining

$$R = [\alpha_0 \mapsto (\alpha_{21} \times \alpha_{22})\ \texttt{event}\ \beta_2].$$

Then $R\,t_1 = t_1 = (\alpha_{11} \times \alpha_{12})$ `event` $\beta_1$ and $R\,\alpha_0 = (\alpha_{21} \times \alpha_{22})$ `event` $\beta_2$ and these types intuitively have the same shape. Returning to Figure 4.3 we would thus expect $\mathcal{M}(\alpha_0, t_1, \sim, R, \sim)$.

If instead we had considered the constraint $\alpha$ `event` $\beta \subseteq \alpha$ then the above procedure would not lead to a matching constraint. We would get

$$R = [\alpha \mapsto \alpha'\ \texttt{event}\ \beta']$$

and the constraint $R\,(\alpha\ \texttt{event}\ \beta) \subseteq R\,\alpha$ then is

$$(\alpha'\ \texttt{event}\ \beta')\ \texttt{event}\ \beta \subseteq \alpha'\ \texttt{event}\ \beta'$$

which does not match; indeed it would seem that matching could go on forever without ever producing a matching result. To detect this situation we have an "occur check": when $\mathcal{M}(\alpha, t, \sim, R, \sim')$ holds no variable in $Dom(R)$ must occur in $t$. This condition fails when $t = \alpha$ `event` $\beta$.

However, there are more subtle ways in which termination may fail. Consider the constraint set

$$\{\alpha_1\ \texttt{event}\ \beta_1 \subseteq \alpha_0,\ \alpha_0 \subseteq \alpha_1\}$$

where only the first constraint does not match. Attempting a match we get

$$R_1 = [\alpha_0 \mapsto \alpha_2\ \texttt{event}\ \beta_2]$$

and note that the "occur check" succeeds. The resulting constraint set is

$$\{\alpha_1\ \texttt{event}\ \beta_1 \subseteq \alpha_2\ \texttt{event}\ \beta_2,\ \alpha_2\ \texttt{event}\ \beta_2 \subseteq \alpha_1\}$$

which may be reduced to

$$\{\alpha_1 \subseteq \alpha_2, \beta_1 \subseteq \beta_2, \alpha_2 \texttt{ event } \beta_2 \subseteq \alpha_1\}.$$

The type part is isomorphic to the initial constraints, so this process may continue forever: we perform a second match and produce a second substitution $R_2$, etc.

To detect this situation we follow [6] in making use of the equivalence relation $\sim$ and extend it with $\alpha_1 \sim \alpha_2$ after the first match that produced $R_1$; the intuition is that $\alpha_1$ and $\alpha_2$ eventually must be bound to types having the same shape. When performing the second match we then require $R_2$ not only to expand $\alpha_1$ but also all $\alpha'$ satisfying $\alpha' \sim \alpha_1$; this means that $R_2$ must expand also $\alpha_2$. Consequently the "extended occur check" $Dom(R_2) \cap FV(\alpha_2 \texttt{ event } \beta_2) = \emptyset$ fails. □

**Remark** Matching bears certain similarities to unification and can actually be defined in terms of unification. In [12] matching is performed by first doing unification and then the resulting substitution is transformed such that it "maps into fresh variables". In [25, Fig. 3.7] it is first checked whether it is possible to unify a certain set of equations, derived from the constraint set; if this is the case then the algorithm behaves similar to the one presented here except that the equivalence relation is no longer needed. □

To formalise the intuition gained from the example we need to be more precise about the shape of a type.

**Definition 4.7** A shape $sh$ is a type with holes in it for all (type or behaviour or region) variables; it may be formally defined by:

$$\begin{array}{rcl} sh & ::= & [\,] \mid \texttt{unit} \mid \texttt{bool} \mid \texttt{int} \mid sh_1 \rightarrow sh_2 \mid sh_1 \rightarrow^{[\,]} sh_2 \\ & \mid & sh_1 \times sh_2 \mid sh \texttt{ list} \mid sh \texttt{ chan } [\,] \mid sh \texttt{ event } [\,] \end{array}$$

We write $sh[\vec{t}, \vec{\beta}, \vec{\rho}]$ for the type obtained by replacing all type holes with the relevant type in the list $\vec{t}$ and replacing all behaviour holes with the relevant behaviour variable in the list $\vec{\beta}$ and replacing all region holes with the relevant region variable in the list $\vec{\rho}$; we assume throughout that the lengths of the lists equal the number of holes and shall dispense with a formal definition. □

$\mathcal{M}(\alpha, t, \sim, R, \sim')$ holds
if $\{\alpha_1, \cdots, \alpha_n\} \cap FV(t) = \emptyset$
and $R = [\alpha_1 \mapsto sh[\vec{\alpha}_1, \vec{\beta}_1, \vec{\rho}_1], \cdots, \alpha_n \mapsto sh[\vec{\alpha}_n, \vec{\beta}_n, \vec{\rho}_n]]$
and $\sim'$ is the least equivalence relation containing the pairs
$\{(\alpha', \alpha'') \mid \alpha' \sim \alpha'' \wedge \{\alpha', \alpha''\} \cap \{\alpha_1, \cdots, \alpha_n\} = \emptyset\} \cup$
$\{(\alpha_{0j}, \alpha_{ij}) \mid \vec{\alpha}_0 = \alpha_{01} \cdots \alpha_{0m}, \vec{\alpha}_i = \alpha_{i1} \cdots \alpha_{im}, 1 \leq i \leq n, 1 \leq j \leq m\}$
where $\{\alpha_1, \cdots, \alpha_n\} = \{\alpha' \mid \alpha' \sim \alpha\}$
and $sh[\vec{\alpha}_0, \vec{\beta}_0, \vec{\rho}_0] = t$ with $\vec{\alpha}_0$ having length $m$
and $\vec{\alpha}_1, \cdots, \vec{\alpha}_n$ are vectors of fresh variables, each of length $m$
and $\vec{\beta}_1, \cdots, \vec{\beta}_n$ are vectors of fresh variables of the same length as $\vec{\beta}_0$
and $\vec{\rho}_1, \cdots, \vec{\rho}_n$ are vectors of fresh variables of the same length as $\vec{\rho}_0$

Figure 4.4: Forced matching

For each type $t$ there clearly exists unique $sh$, $\vec{\alpha}$, $\vec{\beta}$, and $\vec{\rho}$ such that $sh[\vec{\alpha}, \vec{\beta}, \vec{\rho}] = t$.

**Example 4.8** If $sh = ([\,] \times [\,])$ `event` $[\,]$ then $sh[\vec{t}, \vec{\beta}, \vec{\rho}] = (t_1 \times t_2)$ `event` $\beta_1$ if and only if $\vec{t} = t_1 t_2$ and $\vec{\beta} = \beta_1$ and $\vec{\rho} = ()$. □

As already mentioned, the axioms (mr) and (ml) from Fig. 4.3 force a type $t$ to match a type variable $\alpha$ and employ the predicate $\mathcal{M}$ defined in Figure 4.4. This predicate may also be considered a partial function with its first three parameters being input and the last two being output; the "call" $\mathcal{M}(\alpha, t, \sim, R, \sim')$ produces the substitution $R$ and modifies the equivalence relation $\sim$ (over the free type variables of a constraint set $C'$) to another equivalence relation $\sim'$ (over the free type variables of the constraint set $R\,C'$). In axioms (mr) and (ml) the newly produced substitution $R$ is composed with the previously produced substitution. Also note that the "extended occur check" in Figure 4.4 ensures that $R\,t = t$.

**Fact 4.9** Suppose $(S, C, \sim) \Leftrightarrow\!\!\!\!\rightarrow (S', C', \sim')$. Then there exists $R$ such that $S' = R\,S$ and such that $R\,C \rightharpoonup^* C'$. Moreover, if $C$ is well-formed then also $C'$ is well-formed. □

**Remark: type cycles become behaviour cycles.** To understand why $\mathcal{F}$ does not report failure in *more* cases than a "classical type checker", the following example is helpful. Consider the "constraint"

$$C = \{\texttt{int} \rightarrow^{\alpha\ \mathrm{CHAN}\ \rho}\ \texttt{int} \subseteq \alpha\}$$

which will not cause a classical type checker to fail since $\alpha$ is simply unified with $\texttt{int} \rightarrow \texttt{int}$. Now let us see how $\mathcal{F}$ behaves on $C$, when "encoded" into our format:

$$\{\texttt{int} \rightarrow^{\beta}\ \texttt{int} \subseteq \alpha,\ \{\alpha\ \mathrm{CHAN}\ \rho\} \subseteq \beta\}.$$

Here case (mr) in Figure 4.3 is enabled, and consequentially a substitution which maps $\alpha$ into $\texttt{int} \rightarrow^{\beta'}\ \texttt{int}$ (with $\beta'$ new) is applied to the constraints. The resulting constraint set is

$$\{\texttt{int} \rightarrow^{\beta}\ \texttt{int} \subseteq \texttt{int} \rightarrow^{\beta'}\ \texttt{int},\ \{(\texttt{int} \rightarrow^{\beta'}\ \texttt{int})\ \mathrm{CHAN}\ \rho\} \subseteq \beta\}$$

and after applying (dc) twice we end up with the constraint set

$$C' = \{\beta \subseteq \beta',\ \{(\texttt{int} \rightarrow^{\beta'}\ \texttt{int})\ \mathrm{CHAN}\ \rho\} \subseteq \beta\}$$

which cannot be rewritten further. The set $C'$ is atomic so Algorithm $\mathcal{F}$ succeeds on $C$. □

### 4.3.1 Termination and Soundness of $\mathcal{F}$

Having completed the definition of $\mathcal{M}$, $\Leftrightarrow\!\!\!\rightarrow$ and $\mathcal{F}$ we can state:

**Lemma 4.10** $\mathcal{F}(C)$ always terminates (possibly with failure). Suppose that $\mathcal{F}(C)$ succeeds with result $(S', C')$; then

- if $C$ is well-formed then $C'$ is atomic; and
- $C'$ is determined from $S'\,C$ in the sense that $S'\,C \rightharpoonup^{*} C' \not\rightharpoonup$.

**Proof** We first address termination and for this purpose we (much as in [6]) define an ordering on triples $(S, C, \sim)$ as follows: $(S', C', \sim')$ is less than $(S, C, \sim)$ if *either* the number of equivalence classes in $FV(C')$ wrt. $\sim'$ is less than the number of equivalence classes in $FV(C)$ wrt. $\sim$ *or* these numbers are equal but $C'$ is less than $C$ according to the following definition:

for all $i \geq 0$ let $s_i$ be the number of constraints in $C$ containing $i$ symbols and let $s'_i$ be the number of constraints in $C'$ containing $i$ symbols; then $C'$ is less than $C$ if there exists a $n$ such that $s'_n < s_n$ and such that $s'_i = s_i$ for all $i > n$.

This relation on constraint sets is clearly transitive and it is easy to see that it is also well-founded, hence the (lexicographically defined) ordering on triples is well-founded. Thus it suffices to show that if $(S, C, \sim) \Leftrightarrow\!\!\!\!\rightarrow (S', C', \sim')$ then $(S', C', \sim')$ is less than $(S, C, \sim)$. If the rule (dc) has been applied then $C'$ is less than $C$ (as $n$ in the above definition we can use the number of symbols in the constraint being decomposed) and $\sim' = \sim$. If the rule (mr) or (ml) has been applied then the number of equivalence classes wrt. $\sim$ will decrease as can be seen from the definition of $\mathcal{M}$ in Fig. 4.4: the equivalence class containing $\alpha$ is removed (as this class equals $Dom(R)$ and $C' = R\,C$) and no new classes are added (as all type variables in $Ran(R)$ are put into some existing equivalence class).

We have thus proved termination; it is easy to see that the other claims will follow provided we can show that if

$$(\mathrm{Id}, C, \mathrm{Eq}_C) \Leftrightarrow\!\!\!\!\rightarrow^* (S_n, C_n, \sim_n)$$

then $S_n\,C \rightharpoonup^* C_n$ and if $C$ is well-formed then also $C_n$ is well-formed. We do this by induction on the length of the derivation, where the base case as well as the part concerning well-formedness (where we use Fact 4.9) is trivial. For the inductive step, suppose that

$$(\mathrm{Id}, C, \mathrm{Eq}_C) \Leftrightarrow\!\!\!\!\rightarrow^* (S_n, C_n, \sim_n) \Leftrightarrow\!\!\!\!\rightarrow (S_{n+1}, C_{n+1}, \sim_{n+1})$$

where the induction hypothesis ensures that $S_n\,C \rightharpoonup^* C_n$. By Fact 4.9 there exists $R$ such that $S_{n+1} = R\,S_n$ and such that $R\,C_n \rightharpoonup^* C_{n+1}$. As it is easy to see that the relation $\rightharpoonup$ is closed under substitution it holds that $R\,S_n\,C \rightharpoonup^* R\,C_n$, hence the claim. □

**Lemma 4.11** $\mathcal{F}$ *is sound*

If $\mathcal{F}(C) = (S', C')$ then $C' \vdash S'\,C$.

**Proof** By Lemma 4.10 we have $S' \, C \rightharpoonup^* C'$, which yields the claim due to Fact 4.5. $\square$

**Remark** By Fact 4.5 we know that $\rightharpoonup$ is confluent but this does not directly carry over to $\Leftrightarrow\!\rightarrow$ or $\mathcal{F}$: the constraint $\alpha_1 \subseteq \alpha_2$ may yield $([\alpha_1 \mapsto \alpha_0], \{\alpha_0 \subseteq \alpha_2\})$ as well as $([\alpha_2 \mapsto \alpha_0], \{\alpha_1 \subseteq \alpha_0\})$. However, Lemma 4.10 tells us that $\mathcal{F}(C) = (S', C')$ ensures that $C'$ is determined from $S' \, C$; and we conjecture that $S'$ is determined, up to some notion of renaming, from $C$. $\square$

## 4.4 Algorithm $\mathcal{R}$

The purpose of (the optional and somewhat open-ended) algorithm $\mathcal{R}$ is to reduce the size of a constraint set which is already atomic. The techniques used are basically those of [26] and [5], adapted to our framework.

The transformation $\mathcal{R}$ may be described as a non-deterministic rewriting process, operating over triples of the form $(C, t, b)$ with $C$ atomic, and with respect to a fixed environment $A$. We then define $\mathcal{R}$ by:

$$\begin{aligned} \mathcal{R}(C, t, b, A) = {} & \text{let } (C', t', b') \text{ be given by} \\ & \qquad A \vdash (C, t, b) \Leftrightarrow\!\rightarrow^* (C', t', b') \not\Leftrightarrow\!\rightarrow \\ & \text{in } (C', t', b') \end{aligned}$$

The rewriting relation is defined by the axioms of Figure 4.5 and will be explained below (recall that $\dot\cup$ means disjoint union). To understand the axioms, it is helpful to view the constraints as a directed graph where the nodes are either *(i)* type or behaviour or region variables, or *(ii)* non-variable behaviours or channel labels; as the constraints are well-formed, the arrows always have a variable node as the source. With this in mind we define:

**Definition 4.12** We write $(\gamma \Leftarrow^* \gamma') \in C$ if there is a path from $\gamma'$ to $\gamma$; that is if there exists $\gamma_0 \cdots \gamma_n$ $(n \geq 0)$ such that $\gamma_0 = \gamma$ and $\gamma_n = \gamma'$ and $(\gamma_i \subseteq \gamma_{i+1}) \in C$ for all $i \in \{0 \cdots n \Leftrightarrow 1\}$. $\square$

Notice that $(\gamma \Leftarrow^* \gamma) \in C$ holds also if $\gamma \notin FV(C)$. From reflexivity and transitivity of $\subseteq$ we have:

(redund) $A \vdash (C \dot{\cup} \{\gamma' \subseteq \gamma\}, t, b) \Leftrightarrow\!\!\!\rightarrow (C, t, b)$
provided $(\gamma' \Leftarrow^* \gamma) \in C$

(cycle) $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (S\,C, S\,t, S\,b)$
where $S = [\gamma \mapsto \gamma']$ with $\gamma \neq \gamma'$
provided $(\gamma \Leftarrow^* \gamma') \in C$ and $(\gamma' \Leftarrow^* \gamma) \in C$ and
provided $\gamma \notin FV(A) \cup \mathit{ChanVar}(t, b, C)$

(shrink) $A \vdash (C \dot{\cup} \{\gamma' \subseteq \gamma\}, t, b) \Leftrightarrow\!\!\!\rightarrow (S\,C, S\,t, S\,b)$
where $S = [\gamma \mapsto \gamma']$ with $\gamma \neq \gamma'$
provided $\gamma \notin FV(\mathit{RHS}(C), A)$ and
provided $t$, $b$, and each element in $\mathit{LHS}(C)$ is monotonic in $\gamma$

(boost) $A \vdash (C \dot{\cup} \{\gamma \subseteq \gamma'\}, t, b) \Leftrightarrow\!\!\!\rightarrow (S\,C, S\,t, S\,b)$
where $S = [\gamma \mapsto \gamma']$ with $\gamma \neq \gamma'$
provided $\gamma \notin FV(A)$ and
provided $t$, $b$ and each element in $\mathit{LHS}(C)$ is anti-monotonic in $\gamma$

Figure 4.5: Eliminating constraints

**Fact 4.13** If $(\gamma \Leftarrow^* \gamma') \in C$ then also $C \vdash \gamma \subseteq \gamma'$. □

We have a substitution result similar to Lemma 2.18:

**Fact 4.14** Let $S$ be a substitution mapping variables into variables, and suppose $(\gamma \Leftarrow^* \gamma') \in C$. Then also $(S\,\gamma \Leftarrow^* S\,\gamma') \in S\,C$. □

We say that $C$ is cyclic if there exists $\gamma_1, \gamma_2 \in FV(C)$ with $\gamma_1 \neq \gamma_2$ such that $(\gamma_1 \Leftarrow^* \gamma_2) \in C$ and $(\gamma_2 \Leftarrow^* \gamma_1) \in C$.

We now explain the rules: (redund) removes constraints which are redundant due to the ordering $\subseteq$ being reflexive and transitive; applying this rule repeatedly is called "transitive reduction" in [26] and is essential for a compact representation of the constraints.

The remaining rules all replace some variable $\gamma$ by another variable $\gamma'$. However, a "solution" to $C$ will not necessarily map $\gamma$ and $\gamma'$ into identical "terms", and therefore we should not (in the style of $\mathcal{F}$) return the substitution

$[\gamma \mapsto \gamma']$ and subsequently apply it to $A$. In order to maintain soundness (cf. Lemma 4.25) we must therefore demand that $\gamma \notin FV(A)$.

The rule (cycle) collapses cycles in the graph; however, it is not possible to eliminate a cycle which involves *two* elements of $FV(A) \cup \mathit{ChanVar}(t, b, C)$ where $\mathit{ChanVar}()$ is the set of variables occurring inside some (sub)behaviour of the form $t'$ CHAN $\rho$, $\rho\,!\,t'$, or $\rho\,?\,t'$. The requirement concerning $FV(A)$ is due to the remark above; the requirement concerning $\mathit{ChanVar}(t, b, C)$ is needed for technical reasons but notice that we may expect that all variables in $\mathit{ChanVar}(t, b, C)$ will belong to $FV(A')$ for some $A'$ encountered during the algorithm (inside some channel type $t'$ `chan` $\rho$). (In [26] it holds that $\emptyset \vdash t_1 \equiv t_2$ implies $t_1 = t_2$ so if $\gamma$ and $\gamma'$ belong to the same cycle in $C$ then all substitutions that solve $C$ can be written on the form $S'\,[\gamma \mapsto \gamma']$, hence cycle elimination can be part of the analogue of $\mathcal{F}$.)

The rule (shrink) expresses that a variable $\gamma$ can be replaced by its "immediate predecessor" $\gamma'$, and due to the ability to perform transitive reduction this can be strengthened to the requirement that $\gamma'$ is the "only predecessor" of $\gamma$, which can be formalised as the side condition $\gamma \notin FV(\mathit{RHS}(C))$ where $\mathit{RHS}(C) = \{\gamma \mid \exists g : (g \subseteq \gamma) \in C\}$. We can allow $\gamma$ to belong to $t$ and $b$ and $\mathit{LHS}(C)$, where $\mathit{LHS}(C) = \{g \mid \exists \gamma : (g \subseteq \gamma) \in C\}$, as long as we do not "lose instances", that is we must have that $S\,t \subseteq t$, $S\,b \subseteq b$, and $S\,g \subseteq g$ for each $g \in \mathit{LHS}(C)$. This will be the case provided $t$ and $b$ and each element of $\mathit{LHS}(C)$ are *monotonic* in $\gamma$, where for example $t = \alpha_1 \ \to^{\beta_1} \ \alpha_2 \ \to^{\beta_1} \ \alpha_1$ is monotonic in $\gamma$ for *all* $\gamma \notin \{\alpha_1, \alpha_2\}$. A more formal treatment of the concept of monotonicity will be given shortly, for now notice that if $\gamma \notin FV(g)$ or if $g = \gamma$ then $g$ is monotonic in $\gamma$.

The rule (boost) expresses that a variable $\gamma$ can be replaced by its "immediate successor" $\gamma'$, and due to the ability to perform transitive reduction this can be strengthened to the requirement that $\gamma'$ is the "only successor" of $\gamma$. In addition we must demand that we do not "lose instances", that is we must have that $S\,t \subseteq t$, $S\,b \subseteq b$, and $S\,g \subseteq g$ for each $g \in \mathit{LHS}(C)$. This will be the case provided $t$ and $b$ and each element of $\mathit{LHS}(C)$ are *anti-monotonic* in $\gamma$, where for example $t = \alpha_1 \ \to^{\beta_1} \ \alpha_2 \ \to^{\beta_1} \ \alpha_1$ is anti-monotonic in $\gamma$ for *all* $\gamma \notin \{\alpha_1, \beta_1\}$. Notice that if each element of $\mathit{LHS}(C)$ is anti-monotonic in $\gamma$ then $\gamma'$ is in fact the only successor of $\gamma$.

## Monotonicity

**Definition 4.15** Given a constraint set $C$. We say that a substitution $S$ is increasing (respectively decreasing) wrt. $C$ if for all $\gamma$ we have $C \vdash \gamma \subseteq S\,\gamma$ (respectively $C \vdash S\,\gamma \subseteq \gamma$).

We say that a substitution $S$ increases (respectively decreases) $g$ wrt. $C$ whenever $C \vdash g \subseteq S\,g$ (respectively $C \vdash S\,g \subseteq g$). □

We want to define the concepts of monotonicity and anti-monotonicity such that the following result holds:

**Lemma 4.16** Suppose that $g$ is monotonic in all $\gamma \in \mathit{Dom}(S)$; then if $S$ is increasing (respectively decreasing) wrt. $C$ then $S$ increases (respectively decreases) $g$ wrt. $C$.

Suppose that $g$ is anti-monotonic in all $\gamma \in \mathit{Dom}(S)$; then if $S$ is increasing (respectively decreasing) wrt. $C$ then $S$ decreases (respectively increases) $g$ wrt. $C$. □

To this end we make the following definition

**Definition 4.17** We say that $g$ is monotonic in $\gamma$ if $\gamma \in M(g)$; and we say that $g$ is anti-monotonic in $\gamma$ if $\gamma \in A(g)$.

Here the sets $M(g)$ and $A(g)$ are recursively defined below, where $\mathcal{V}$ denotes the "universe" of variables:

$$
\begin{aligned}
&M(\gamma) = \mathcal{V} \text{ and } A(\gamma) = \mathcal{V} \setminus \{\gamma\};\\
&M(\texttt{unit}) = M(\texttt{bool}) = M(\texttt{int}) = M(\varepsilon) = M(\{l\}) = \mathcal{V};\\
&A(\texttt{unit}) = A(\texttt{bool}) = A(\texttt{int}) = A(\varepsilon) = A(\{l\}) = \mathcal{V};\\
&M(t_1 \rightarrow t_2) = A(t_1) \cap M(t_2);\\
&A(t_1 \rightarrow t_2) = M(t_1) \cap A(t_2);\\
&M(t_1 \rightarrow^{\beta} t_2) = A(t_1) \cap M(t_2);\\
&A(t_1 \rightarrow^{\beta} t_2) = (M(t_1) \cap A(t_2)) \setminus \{\beta\};\\
&M(t_1 \times t_2) = M(t_1) \cap M(t_2);\\
&A(t_1 \times t_2) = A(t_1) \cap A(t_2);\\
&M(t\ \texttt{list}) = M(t) \text{ and } A(t\ \texttt{list}) = A(t);\\
&M(t\ \texttt{chan}\ \rho) = \mathcal{V} \setminus FV(t);\\
&A(t\ \texttt{chan}\ \rho) = \mathcal{V} \setminus (\{\rho\} \cup FV(t));\\
&M(t\ \texttt{event}\ \beta) = M(t);
\end{aligned}
$$

$A(t\ \texttt{event}\ \beta) = A(t) \setminus \{\beta\}$ ;
$M(b_1; b_2) = M(b_1 + b_2) = M(b_1) \cap M(b_2)$;
$A(b_1; b_2) = A(b_1 + b_2) = A(b_1) \cap A(b_2)$;
$M(\mathit{SPAWN}\, b) = M(b)$ and $A(\mathit{SPAWN}\, b) = A(b)$;
$M(t\ \text{CHAN}\ \rho) = M(\rho\,!\,t) = M(\rho\,?\,t) = \mathcal{V} \setminus (\{\rho\} \cup FV(t))$;
$A(t\ \text{CHAN}\ \rho) = A(\rho\,!\,t) = A(\rho\,?\,t) = \mathcal{V} \setminus (\{\rho\} \cup FV(t))$.

**Fact 4.18** For all types/behaviours/regions $g$, it holds that $M(g) \cap A(g) = \mathcal{V} \setminus FV(g)$ and that $(M(g) \cup A(g)) \cap \mathit{ChanVar}(g) = \emptyset$. (So if $g$ is monotonic as well as anti-monotonic in $\gamma$, then $\gamma \notin FV(g)$.)

For all behaviours $b$, $A(b) = \mathcal{V} \setminus FV(b)$. □

Now we can prove Lemma 4.16:

**Proof** Induction on $g$, we list some typical cases:

**$g$ is a variable:** The claims follow from the fact that if $g$ is anti-monotonic in all $\gamma \in \mathit{Dom}(S)$, then $g \notin \mathit{Dom}(S)$.

**$g$ is a function type $t_1 \rightarrow^\beta t_2$:** First consider the sub-case where $g$ is monotonic in all $\gamma \in \mathit{Dom}(S)$ and where $S$ is increasing wrt. $C$. Then $\gamma \in \mathit{Dom}(S)$ gives $\gamma \in M(t_1 \rightarrow^\beta t_2)$, and we infer that $\gamma \in A(t_1)$ and $\gamma \in M(t_2)$, so that $t_1$ is anti-monotonic in $\gamma$ whereas $t_2$ is monotonic in $\gamma$. We can thus apply the induction hypothesis to infer that $S$ decreases $t_1$ wrt. $C$ and that $S$ increases $t_2$ wrt. $C$. But then it is straightforward (as $C \vdash \beta \subseteq S\,\beta$) that $S$ increases $g$ wrt. $C$.

The other sub-cases are rather similar.

**$g$ is a sequential behaviour $b_1; b_2$:** First consider the sub-case where $g$ is anti-monotonic in all $\gamma \in \mathit{Dom}(S)$ and where $S$ is increasing wrt. $C$. Then $\gamma \in \mathit{Dom}(S)$ gives $\gamma \in A(b_1; b_2)$, and we infer that $\gamma \in A(b_1)$ and $\gamma \in A(b_2)$, so that $b_1$ and $b_2$ are both anti-monotonic in $\gamma$. We can thus apply the induction hypothesis to infer that $S$ decreases $b_1$ as well as $b_2$ wrt. $C$. But then it is straightforward that $S$ decreases $g$ wrt. $C$.

The other sub-cases are similar.

$g$ **is a channel behaviour** $t$ CHAN $\rho$**:** First consider the sub-case where $g$ is monotonic in all $\gamma \in Dom(S)$. Then $\gamma \in Dom(S)$ gives $\gamma \in M(t \text{ CHAN } \rho)$, that is $\gamma \notin FV(t)$ and $\gamma \neq \rho$. Thus $S\,t = t$ and $S\,\rho = \rho$, so clearly $S$ increases as well as decreases $g$ wrt. $C$.

The other sub-case is similar. □

**Example 4.19** Let $C$ and $t$ be given by

$$C = \{\alpha_1 \subseteq \alpha_2\} \text{ and } t = \alpha_1 \rightarrow^{\beta} \alpha_2. \tag{1}$$

As $t$ is monotonic in $\alpha_2$, it is possible to apply (shrink) and get

$$C' = \emptyset \text{ and } t' = \alpha_1 \rightarrow^{\beta} \alpha_1. \tag{2}$$

The soundness and completeness of this transformation may informally be argued as follows: (1) "denotes" the set of types

$$\{t_1 \rightarrow^{\beta} t_2 \mid \emptyset \vdash t_1 \subseteq t_2\}$$

but this is also the set of types denoted by (2), due to the presence of subtyping.

Notice that since $t$ is anti-monotonic in $\alpha_1$, it is also possible to apply (boost) from (1) and arrive at

$$C' = \emptyset \text{ and } t' = \alpha_2 \rightarrow^{\beta} \alpha_2$$

which modulo renaming is equal to (2). □

**Example 4.20** Let $C$ and $t$ be given by

$$C = \{\alpha_2 \subseteq \alpha_1\} \text{ and } t = \alpha_1 \rightarrow^{\beta} \alpha_2.$$

Then neither (shrink) nor (boost) is applicable, as $t$ is not monotonic in $\alpha_1$ nor anti-monotonic in $\alpha_2$. □

**Observation 4.21** The rules in Fig. 4.5 might be brought to a more symmetric form (employing that all right hand sides of constraints are assumed to be variables):

- for the rule (shrink), the requirement $\gamma \notin FV(RHS(C))$ can be replaced by the requirement that each element of $RHS(C)$ is anti-monotonic in $\gamma$;
- for the rule (boost), one can add the (void) requirement that each element of $RHS(C)$ is monotonic in $\gamma$;
- for the rules (shrink) and (boost), one can add the requirement that $\gamma \notin ChanVar(t, b, C)$ (which follows from the other requirements, using Fact 4.18).

### 4.4.1 Termination and Soundness of $\mathcal{R}$

**Lemma 4.22** $\mathcal{R}$ always terminates. If $\mathcal{R}(C, t, b, A) = (C', t', b')$ with $C$ atomic then $C'$ is atomic.

**Proof** Termination is ensured since each rewriting step either decreases the number of constraints, or (as is the case for (cycle)) decreases the number of variables without increasing the number of constraints. Each rewriting step trivially preserves atomicity. □

Turning to soundness, we first prove an auxiliary result about the rewriting relation:

**Lemma 4.23** Suppose $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ with $C$ atomic. Then there exists $S$ such that $C' \vdash S\,C$, $t' = S\,t$, $b' = S\,b$, and $A = S\,A$.

**Proof** For (redund) we can use $S = \mathrm{Id}$ and the claim follows from Fact 4.13. For (cycle) the claim is trivial; and for (shrink) and (boost) the claim follows from the fact that with $(\gamma_1 \subseteq \gamma_2)$ the "discarded" constraint it holds that $(S\,\gamma_1 \subseteq S\,\gamma_2)$ is an instance of reflexivity. □

Using Lemma 2.18 and Lemma 2.19 we then get:

**Corollary 4.24** Suppose $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ with $C$ atomic.
If $C, A \vdash e : t \,\&\, b$ then $C', A \vdash e : t' \,\&\, b'$ (and with the same shape). □

By repeated application of this corollary we get the desired result:

**Lemma 4.25** Suppose that $\mathcal{R}(C, t, b, A) = (C', t', b')$ with $C$ atomic. If $C, A \vdash e : t \,\&\, b$ then $C', A \vdash e : t' \,\&\, b'$ (and with the same shape). $\square$

### 4.4.2 Variants of $\mathcal{R}$

It is crucial for the use of $\mathcal{R}$ that Lemma 4.25 as well as Lemma 4.22 hold. This will be the case for $\mathcal{R}$ trivially defined by $\mathcal{R}(C, t, b, A) = (C, t, b)$, but one may also consider more powerful variants where the set of rewritings presented in Figure 4.5 is augmented with other rules (all satisfying Corollary 4.24). It will be natural to allow the replacement of $b$ by a "smaller" behaviour $b'$ provided that $\emptyset \vdash b \equiv b'$ holds (then say $\varepsilon; \beta; \varepsilon$ can be replaced by $\beta$).

### 4.4.3 Results concerning Confluence and Determinism

For $\mathcal{R}$ as defined by Fig. 4.5, we have the following result showing that no new paths are introduced in the graph:

**Lemma 4.26** Suppose $A \vdash (C', t', b') \Leftrightarrow\!\!\!\rightarrow (C'', t'', b'')$ and $\gamma_1, \gamma_2 \in FV(C'')$. Then $(\gamma_1 \Leftarrow^* \gamma_2) \in C'$ holds iff $(\gamma_1 \Leftarrow^* \gamma_2) \in C''$ holds.

**Proof** See Appendix C. $\square$

It is easy to see (using Observation 4.21) that if $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ then $\mathit{ChanVar}(t, b, C) = \mathit{ChanVar}(t', b', C')$, yielding the following

**Observation 4.27** Suppose $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ where the rule (cycle) is not applicable from the configuration $(C, t, b)$. Then the rule (cycle) is not applicable from the configuration $(C', t', b')$ either. $\square$

This suggests that an implementation could begin by collapsing all cycles once and for all, without having to worry about cycles again. On the other hand, it is not possible to perform transitive reduction in a separate phase as (redund) may become enabled after applying (shrink) or (boost): as an example consider the situation where $C$ contains the constraints

$$\begin{array}{ll} \gamma_0 \subseteq \gamma, & \gamma \subseteq \gamma_1, \\ \gamma_0 \subseteq \gamma', & \gamma' \subseteq \gamma_1 \end{array}$$

and (redund) is not applicable. By applying (shrink) with the substitution $[\gamma \mapsto \gamma_0]$ we end up with the constraints

$$\gamma_0 \subseteq \gamma_1, \gamma_0 \subseteq \gamma', \gamma' \subseteq \gamma_1$$

of which the former can be eliminated by (redund).

Concerning confluency, one would like to show a "diamond property" but this cannot be done in the presence of cycles in the constraint set (especially if these contain multiple elements of $FV(A)$): as an example consider the constraints

$$\gamma_0 \subseteq \gamma, \gamma_0 \subseteq \gamma', \gamma \subseteq \gamma', \gamma' \subseteq \gamma$$

with $\gamma, \gamma' \in FV(A)$; here we can apply (redund) to eliminate either the first or the second constraint but then we are stuck as (cycle) is not applicable and therefore we cannot complete the diamond. As another example, consider the case where we have a cycle containing $\gamma_0$, $\gamma_1$ and $\gamma_2$ with $\gamma_0, \gamma_1 \in FV(A)$. Then we can apply (cycle) to map $\gamma_2$ into either $\gamma_0$ or $\gamma_1$ but then we are stuck and the graphs will be different (due to the arrows to or from $\gamma_2$) unless we devise some notion of graph equivalence.

On the other hand, we have the following result:

**Proposition 4.28** Suppose that

$$A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C_1, t_1, b_1) \text{ and}$$
$$A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C_2, t_2, b_2)$$

where $C$ is *acyclic* as well as atomic. Then there exists $(C_1', t_1', b_1')$ and $(C_2', t_2', b_2')$, which are equal up to renaming, such that

$$A \vdash (C_1, t_1, b_1) \Leftrightarrow\!\!\!\rightarrow^{\leq 1} (C_1', t_1', b_1') \text{ and}$$
$$A \vdash (C_2, t_2, b_2) \Leftrightarrow\!\!\!\rightarrow^{\leq 1} (C_2', t_2', b_2').$$

(Here "$\Leftrightarrow\!\!\!\rightarrow^{\leq 1}$" denotes "=" or "$\Leftrightarrow\!\!\!\rightarrow$".)

**Proof** See Appendix C. □

## 4.5 Syntactic Soundness of Algorithm $\mathcal{W}$

The algorithm $\mathcal{W}$ always terminates and maintains certain invariants:

**Lemma 4.29** $\mathcal{W}(A, e)$ and $\mathcal{W}'(A, e)$ always terminate (possibly with failure). If $A$ is well-formed then the following holds:

- if $\mathcal{W}(A, e) = (S, t, b, C)$ then $C$ is atomic;
- if $\mathcal{W}'(A, e) = (S, t, b, C)$ then $C$ is well-formed;
- all subcalls to $\mathcal{W}$ and $\mathcal{W}'$ are made with an environment which is well-formed.

**Proof** This result is proved by structural induction in $e$; for `let` we use Fact 4.4; for $\mathcal{F}$ and $\mathcal{R}$ we employ Lemma 4.10 and Lemma 4.22. □

Note that if the expression $e$ only mentions identifiers in the domain of $A$ (as when $e$ is a source program), and if $e$ only mentions constants in the domain of $A$, then the only possible form for failure is due to $\mathcal{F}$. In Sect. 5.6 we shall see that then also ML typing would have failed.

As a final preparation for establishing soundness of algorithm $\mathcal{W}$ we establish a result about our formula for generalisation.

**Lemma 4.30** Let $C$ be atomic; then

$$C, A \vdash_n e : t \,\&\, b \text{ implies } C, A \vdash_n e : \mathit{GEN}(A, b)(C, t) \,\&\, b.$$

**Proof** See Appendix C. □

**Theorem 4.31** If $\mathcal{W}(A, e) = (S, t, b, C)$ with $A$ well-formed and $e \in \mathit{EExp}$, then $C, S\,A \vdash_n e : t \,\&\, b$.

**Proof** The result is shown by induction in $e$ with a similar result for $\mathcal{W}'$. See Appendix C for the details. □

## 4.6 Relation to ML Typing

We shall now see that if $\mathcal{W}$ succeeds on some sequential source program $e \in \mathit{Exp}$ then $e$ is "ML-typeable".

Let $A$ be as in Figure 2.4, and suppose that $\mathcal{W}(A, e)$ succeeds with result $(S, t, b, C)$. As $A$ is closed and well-formed (Fact 2.16), it by Theorem 4.31 holds that

$$C, A \vdash_n e : t \,\&\, b$$

where $C$ is atomic by Lemma 4.29. Let $S'$ unify all type variables, then we by Lemmas 2.18 and 2.19 obtain

$$C', A \vdash_n e : S'\,t \,\&\, S'\,b$$

where $C'$ contains no type constraints. Let $A'$ be the restriction of $A$ to sequential constants, then clearly also

$$C', A' \vdash_n e : S'\,t \,\&\, S'\,b$$

and as $A'$ is $\beta$-sequential, Theorem 2.25 tells us that $e$ can be typed in the ML type system.

# Chapter 5

# Completeness of the Inference Algorithm

## 5.1 Lazy Instance

We now begin the preparations for formulating syntactic completeness of algorithm $\mathcal{W}$, as done in Sect. 5.2; to do so we must adapt the notion of lazy instance from [5].

**Definition 5.1** The type $t$ is a *generic instance* (with respect to $C$) of the type scheme $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$, written $t <_C \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$, if and only if there exists a substitution $S$ such that $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$, $C \vdash S\,C_0$, and $C \vdash S\,t_0 \subseteq t$. □

Notice that $t <_C \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ holds iff there exists $S_0$ such that $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is solvable from $C$ by $S_0$ (cf. Def. 2.3) and $C \vdash S_0\,t_0 \subseteq t$; thanks to the latter feature (subtyping) a type scheme can "represent" a large class of types "lazily".

**Definition 5.2** The type scheme $ts_1$ is a generic instance (with respect to $C$) of the type scheme $ts_2$, written $ts_1 \leq_C ts_2$, if and only if for all $C'$ and $t$: whenever $C' \vdash C$ and $t <_{C'} ts_1$ then also $t <_{C'} ts_2$. □

**Remark.** Note that unlike the corresponding concept in [26] we allow to replace $C$ by any $C'$ such that $C' \vdash C$ thus borrowing ideas from Kripke-semantics. In our view this is *essential* for achieving substitution and entailment properties throughout and for avoiding the problem identified in [26] about enlarging the constraint set. □

We write $\sigma_1 \leq_C \sigma_2$ also in the case where $\sigma_1$ or $\sigma_2$ are types: here $ts \leq_C t$ means $ts \leq_C \forall(() : \emptyset).\ t$ and $t \leq_C \sigma$ means $\forall(() : \emptyset).\ t \leq_C \sigma$. For assumptions $A_1$ and $A_2$ with $Dom(A_1) = Dom(A_2)$ we write $A_1 \leq_C A_2$ if and only if for all entries $\sigma_1$ in $A_1$ it for the corresponding entry $\sigma_2$ in $A_2$ holds that *(i)* $\sigma_2$ is a type scheme iff $\sigma_1$ is a type scheme, and *(ii)* $\sigma_1 \leq_C \sigma_2$.

**Fact 5.3** *Generic Instances and Types*

(a) $t <_C \forall(() : \emptyset).\ t_0$ if and only if $C \vdash t_0 \subseteq t$.

(b) $\forall(() : \emptyset).\ t \leq_C ts$ if and only if $t <_C ts$.

**Proof** Only the "if" part of case (b) is non-trivial. So let $t' <_{C'} \forall(() : \emptyset).\ t$ with $C' \vdash C$, our task is to prove $t' <_{C'} ts$ where we write $ts = \forall(G_1 : C_1).\ t_1$. From $t <_C ts$ we get a substitution $S$ with $Dom(S) \subseteq G_1$ such that $C \vdash S\,C_1$ and $C \vdash S\,t_1 \subseteq t$. From $t' <_{C'} \forall(() : \emptyset).\ t$ and (a) it follows that $C' \vdash t \subseteq t'$. It follows (using Lemma 2.19) that $C' \vdash S\,C_1$ and $C' \vdash S\,t_1 \subseteq t'$ and hence $t' <_{C'} ts$. □

**Lemma 5.4** *Properties of* $\leq_C$

(a) $\leq_C$ is reflexive and transitive.

(b) If $\sigma_1 \leq_C \sigma_2$ and $S$ is a substitution then $S\,\sigma_1 \leq_{S\,C} S\,\sigma_2$.

(c) If $\sigma_1 \leq_C \sigma_2$ and $C' \vdash C$ then $\sigma_1 \leq_{C'} \sigma_2$.

**Proof** See Appendix D. □

We now turn our attention to so-called typing judgements of the form $jdg = C, A \mid e : \sigma \,\&\, b$; these are merely five-tuples written in a more readable form and we write $\vdash jdg$ for $C, A \vdash e : \sigma \,\&\, b$ and $S\,(jdg)$ for $S\,C, S\,A \mid e : S\,\sigma \,\&\, S\,b$.

A judgement is an instance of another judgement if it has a stronger constraint set, a type (scheme) with fewer instances, a larger behaviour, and an environment with more instances; the intuition is that if $jdg_1$ is an instance of $jdg_2$ and $\vdash jdg_2$ then certainly also $\vdash jdg_1$.

**Definition 5.5** A typing judgement $jdg_1 = C_1, A_1 \mid e : \sigma_1 \,\&\, b_1$ is an $S$-*instance* of a typing judgement $jdg_2 = C_2, A_2 \mid e : \sigma_2 \,\&\, b_2$, to be written $jdg_1 \preceq^S jdg_2$, if and only if $C_1 \vdash S\,C_2$, $S\,A_2 \leq_{C_1} A_1$, $\sigma_1 \leq_{C_1} S\,\sigma_2$ and $C_1 \vdash S\,b_2 \subseteq b_1$. □

Note that if $\sigma_1 = t_1$ and $\sigma_2 = t_2$ then by Fact 5.3 the condition $\sigma_1 \leq_{C_1} S\,\sigma_2$ amounts to $C_1 \vdash S\,t_2 \subseteq t_1$.

**Fact 5.6** $jdg_1 \preceq^S jdg_2$ if and only if $jdg_1 \preceq^{\mathrm{Id}} S\,jdg_2$. □

**Lemma 5.7** *Properties of* $\preceq^{Id}$

(a) $\preceq^{\mathrm{Id}}$ is reflexive and transitive.

(b) If $jdg_1 \preceq^{\mathrm{Id}} jdg_2$ and $S$ is a substitution then $S\,jdg_1 \preceq^{\mathrm{Id}} S\,jdg_2$.

(c) If $C_1, A_1 \mid e : \sigma_1 \,\&\, b_1 \preceq^{\mathrm{Id}} jdg_2$ and $C_0 \vdash C_1$ then $C_0, A_1 \mid e : \sigma_1 \,\&\, b_1 \preceq^{\mathrm{Id}} jdg_2$.

**Proof** See Appendix D. □

**Lemma 5.8** *Generalisation Lemma*
If $C^*, A^* \mid e : t^* \,\&\, b^* \preceq^S C, A \mid e : t \,\&\, b$ then
$C^*, A^* \mid e : t^* \,\&\, b^* \preceq^S C, A \mid e : \mathit{GEN}(A, b)(C, t) \,\&\, b$
(where $\mathit{GEN}$ is defined as in Sect. 4.2).

**Proof** See Appendix D. □

## 5.2 The Completeness Result

The notion of lazy instance [5] corresponds to our notion of $S$-instance and is a key tool in the formulation of syntactic completeness (see Theorem 5.18) which allows a proof by induction:

if $C^*, A^* \vdash_n^{at} e : \sigma^* \,\&\, b^*$ and
$C^*$ is atomic and
$A^* \leq_{C^*} S'' A$ with $A$ well-formed
then there exists $S$, $t$, $b$, $C$, and $S'$ such that
$\mathcal{W}(A, e) = (S, t, b, C)$
$S'' \overline{\overline{\,_{NF(A,e)}}}\, S' S$
$C^*, S'' A \mid e : \sigma^* \,\&\, b^* \preceq^{S'} C, S A \mid e : GEN(S A, b)(C, t) \,\&\, b$

Here $S_1 \overline{\overline{\,_X}}\, S_2$ means that $\forall \gamma \in X : S_1 \gamma = S_2 \gamma$ and $NF(A, e)$ is the complement of the set $F(A, e)$ of freshly generated variables during the call $\mathcal{W}(A, e)$; note that $FV(A) \subseteq NF(A, e)$ by the meaning of "freshness". And $C^*, A^* \vdash^{at} e : \sigma^* \,\&\, b^*$ denotes an *atomic inference*; i.e. an inference tree where for each application of the rule (gen) we put the following demand on the type scheme $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ occurring in the conclusion: $C_0$ must be atomic.

As we shall see below (Lemma 5.13) it is the restriction to atomic constraints $C^*$ that allows algorithm $\mathcal{F}$ to manipulate type constraints without losing instances. The decomposition of $S''$ into $S' S$ is standard and may be found also in [26, 8]. Just as in [26] our hypothesis cannot simply be $A^* = S'' A$ but has to be $A^* \leq_{C^*} S'' A$; this is necessary for the inductive proof due to the fact that the occurrences of rule (gen) in $C^*, A^* \vdash e : \sigma^* \,\&\, b^*$ allow to generalise over a smaller set of variables than is forced by the use of *GEN* in algorithm $\mathcal{W}$. Therefore we also have to use $S'' A$ rather than $A^*$ in the final judgement.

Below we shall discuss the severeness of the various restrictions on the completeness result; for that purpose we consider an arbitrary derivation $C^*, A \vdash e : t^* \,\&\, b^*$ with $e$ closed. It will be most natural to require $A$ to behave as in Figure 2.4 (implying well-formedness, cf. Fact 2.16); and it is clearly possible to find atomic constraints $C_0^*$ such that $A$ is solvable from $C_0^*$; thus (Lemma 2.19 and Lemma 2.24) we can in fact assume $C^* \cup C_0^*, A \vdash_n e : t^* \,\&\, b^*$. It is now possible to apply the completeness result (with $A^* = A$ and $S'' = \mathrm{Id}$), *provided that* $C^*$ as well as the inference is atomic. We believe that most inferences which occur in practice will in fact be atomic; unfortunately we have not been able to give a general method for transforming non-atomic inferences into atomic inferences and it is still open whether such a method exists.

The completeness result is thus not quite as general as one might wish; in Sect. 5.6, however, we shall see that for a large class of programs (those which are typeable in ML) algorithm $\mathcal{W}$ does in fact succeed (and produces "the most general typing").

Before proving Theorem 5.18 we must address the completeness of $\mathcal{F}$ and $\mathcal{R}$.

## 5.3 Completeness of $\mathcal{F}$

We first introduce the crucial concept of *matching*:

**Definition 5.9** The types $t_1$ and $t_2$ match, written $t_1 \approx t_2$, if and only if their unique decompositions $t_i = sh_i[\vec{\alpha}_i, \vec{\beta}_i, \vec{\rho}_i]$ satisfy that $sh_1 = sh_2$.

We say that $R$ is a *matching substitution* for a constraint set $C$ whenever $R\,t_1 \approx R\,t_2$ for all $(t_1 \subseteq t_2) \in C$. □

**Fact 5.10** The relation $\approx$ is a congruence on types. □

Moreover, the relation $\approx$ is an "inverse congruence" in the sense that if e.g. $(t_1\ \texttt{event}\ \beta_1) \approx t_2'$ then $t_2' = (t_2\ \texttt{event}\ \beta_2)$ where $t_1 \approx t_2$.

**Fact 5.11** Suppose the type constraints in $C^*$ are all of the form $\alpha_1 \subseteq \alpha_2$. If $C^* \vdash t_1 \subseteq t_2$ then $t_1 \approx t_2$.

**Proof** This is proved by induction on the inference $C^* \vdash t_1 \subseteq t_2$. If $(t_1 \subseteq t_2)$ is an assumption in $C^*$ the result is immediate. The cases of reflexivity and transitivity are immediate because $\approx$ is an equivalence relation. The remaining cases are straightforward applications of the induction hypothesis, using Fact 5.10 and the subsequent remark. □

Algorithm $\mathcal{F}$ produces the most general matching substitution:

**Lemma 5.12** Suppose that $C$ is well-formed and that $R$ is a matching substitution for $C$. Then $\mathcal{F}(C)$ will always succeed, and whenever $\mathcal{F}(C) = (S', C')$ there exists $R'$ such that $R'$ is a matching substitution for $C'$ and $R \underset{NF(C)}{=\!=} R'\,S'$, where $NF(C)$ is the complement of the set $F(C)$ of fresh variables generated in the call $\mathcal{F}(C)$.

If $C$ is well-formed and $C^* \vdash R\,C$ with $C^*$ atomic, then (by Fact 5.11) $R$ is a matching substitution for $C$, and whenever $\mathcal{F}(C)$ succeeds with result $(S', C')$ the substitution $R'$ mentioned in the first part of the lemma can be chosen such that $C^* \vdash R'\,C'$.

**Proof** See Appendix D. □

To highlight the way in which the completeness proof for $\mathcal{W}$ makes use of the completeness of $\mathcal{F}$ we state the following result that is a consequence of Lemma 5.12 and that is more directly applicable in the proof of Theorem 5.18.

**Lemma 5.13** Suppose $jdg^* = C^*, A^* \mid e : t^* \,\&\, b^*$ has $C^*$ to be atomic; and suppose $jdg^* \preceq^R jdg$ where $jdg = C, A \mid e : t \,\&\, b$ with $C$ well-formed. Then there exists $C'$, $S'$ and $R'$ such that $\mathcal{F}(C) = (S', C')$, $R \underset{NF(C)}{=\!=} R'\,S'$, and $jdg^* \preceq^{R'} C', S'\,A \mid e : S'\,t \,\&\, S'\,b$.

**Proof** Let $jdg^*$ and $jdg$ satisfy the conditions stated. Since $C^* \vdash R\,C$ we can use Lemma 5.12 to yield $C'$, $S'$ and $R'$ such that $\mathcal{F}(C) = (S', C')$, $R \underset{NF(C)}{=\!=} R'\,S'$, and $C^* \vdash R'\,C'$. Hence $jdg^* \preceq^R jdg$ may be rewritten as $jdg^* \preceq^{R'\,S'} jdg$ which amounts to

$$jdg^* \preceq^{R'} S'\,C, S'\,A \mid e : S'\,t \,\&\, S'\,b$$

and since $C^* \vdash R'\,C'$ we may replace $S'\,C$ with $C'$ and thus achieve

$$jdg^* \preceq^{R'} C', S'\,A \mid e : S'\,t \,\&\, S'\,b$$

which is the desired result. □

## 5.4 Completeness of $\mathcal{R}$

First an auxiliary result (where we like to drop the condition $Dom(S) \cap ChanVar(g) = \emptyset$, but this cannot be done due to the lack of rules in Fig. 2.7 relating say $\rho\,!\,t$ and $\rho'\,!\,t'$):

**Lemma 5.14** Suppose that $C \vdash \gamma \equiv S\,\gamma$ holds for all $\gamma$; then for $g$ such that $Dom(S) \cap ChanVar(g) = \emptyset$ we also have $C \vdash g \equiv S\,g$.

**Proof** Induction in $g$. If $g$ is a variable, the result follows from the assumptions. If $g$ is of the form $t$ CHAN $\rho$ or $\rho\,!\,t$ or $\rho\,?\,t$, the assumptions tell us that $Dom(S) \cap FV(g) = \emptyset$ which trivially implies $C \vdash g \equiv S\,g$. Otherwise, the induction hypothesis will tell us that for all immediate subcomponents $g_i$ of $g$ it holds that $C \vdash g_i \equiv S\,g_i$; using the laws of Figs. 2.6 and 2.7 this can be combined to yield the desired result. □

Next a crucial result about the rewriting relation:

**Lemma 5.15** Suppose $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ with $C$ atomic.
Then $C \vdash C'$, $C \vdash t' \subseteq t$, and $C \vdash b' \subseteq b$.

**Proof** We use the terminology of Figure 4.5; for (redund) the claim is trivial. For (cycle) the claim follows from the fact that by Lemma 5.14 we for all subcomponents $g$ of $(t, b, C)$ have $C \vdash g \equiv S\,g$; so $C \vdash S\,t \subseteq t$ and $C \vdash S\,b \subseteq b$ and for $(g_1 \subseteq g_2) \in C$ we have $C \vdash S\,g_1 \equiv g_1 \subseteq g_2 \equiv S\,g_2$.

For (shrink), our first task is to show that if $(g \subseteq \gamma_0) \in C$ then

$$C \cup \{\gamma' \subseteq \gamma\} \vdash S\,g \subseteq S\,\gamma_0.$$

But this follows since

- $\{\gamma' \subseteq \gamma\} \vdash S\,g \subseteq g$ (using Lemma 4.16 and the assumption about $LHS(C)$ being monotonic in $\gamma$);
- $C \vdash g \subseteq \gamma_0$;
- $\gamma_0 = S\,\gamma_0$ (by the assumption that $\gamma \notin FV(RHS(C))$).

Next we must show $C \cup \{\gamma' \subseteq \gamma\} \vdash S\,t \subseteq t$ and $C \cup \{\gamma' \subseteq \gamma\} \vdash S\,b \subseteq b$, but this follows from Lemma 4.16 since by assumption it holds that $t$ and $b$ are monotonic in $\gamma$.

For (boost), our first task is to show that if $(g \subseteq \gamma_0) \in C$ then

$$C \cup \{\gamma \subseteq \gamma'\} \vdash S\,g \subseteq S\,\gamma_0.$$

But this follows since

- $\{\gamma \subseteq \gamma'\} \vdash S\,g \subseteq g$ (using Lemma 4.16 and the assumption about *LHS*($C$) being anti-monotonic in $\gamma$);
- $C \vdash g \subseteq \gamma_0$;
- $\{\gamma \subseteq \gamma'\} \vdash \gamma_0 \subseteq S\,\gamma_0$.

Next we must show $C \cup \{\gamma \subseteq \gamma'\} \vdash S\,t \subseteq t$ and $C \cup \{\gamma \subseteq \gamma'\} \vdash S\,b \subseteq b$, but this follows from Lemma 4.16 since by assumption it holds that $t$ and $b$ are anti-monotonic in $\gamma$. □

To highlight the way in which the completeness proof for $\mathcal{W}$ makes use of the completeness of $\mathcal{R}$ we state the following results that are consequences of Lemma 5.15 and that are more directly applicable in the proof of Theorem 5.18.

**Corollary 5.16** Suppose $A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C', t', b')$ with $C$ atomic and suppose $jdg^* \preceq^R jdg$ where $jdg^* = C^*, A^* \mid e : t^* \,\&\, b^*$ and $jdg = C, A \mid e : t \,\&\, b$. Then $jdg^* \preceq^R jdg'$ where $jdg' = C', A \mid e : t' \,\&\, b'$.

**Proof** The situation is that $C^* \vdash R\,C$, $R\,A \leq_{C^*} A^*$, $t^* \leq_{C^*} R\,t$, and $C^* \vdash R\,b \subseteq b^*$. By Lemma 5.15 it holds that $C \vdash C'$, $C \vdash t' \subseteq t$, and $C \vdash b' \subseteq b$. By Lemma 2.18 and Lemma 2.19 we now infer that $C^* \vdash R\,C'$, $C^* \vdash R\,b' \subseteq R\,b$, and $C^* \vdash R\,t' \subseteq R\,t$ which by Fact 5.3 amounts to $R\,t \leq_{C^*} R\,t'$.

Our task is to show that $C^* \vdash R\,C'$, $R\,A \leq_{C^*} A^*$, $t^* \leq_{C^*} R\,t'$, and $C^* \vdash R\,b' \subseteq b^*$. All this follows easily from what is shown above. □

By repeated application of Corollary 5.16 (and using Lemma 4.22) we get the desired result:

**Lemma 5.17** Suppose $jdg^* \preceq^R jdg$ with $jdg = C, A \mid e : t \,\&\, b$ where $C$ is atomic. Then $\mathcal{R}(C, t, b, A)$ will always succeed, and whenever $\mathcal{R}(C, t, b, A) = (C', t', b')$ it holds that $jdg^* \preceq^R jdg'$ where $jdg' = C', A \mid e : t' \,\&\, b'$. □

### 5.4.1 Variants of $\mathcal{R}$

In Sect. 4.4.2 we considered alternative versions of $\mathcal{R}$; for each of these we must check that Lemma 5.17 still holds. This is trivial for $\mathcal{R}$ defined by $\mathcal{R}(C,t,b,A) = (C,t,b)$; and we can also allow to augment Fig. 4.5 with a rewriting step that replaces $b$ by a "smaller" behaviour $b'$ where $\emptyset \vdash b \equiv b'$, as Corollary 5.16 can still be established.

## 5.5 Completeness of Algorithm $\mathcal{W}$

**Theorem 5.18** *Completeness Theorem*

If $C^*, A^* \vdash_n^{at} e : \sigma^* \,\&\, b^*$ and
$C^*$ is atomic and
$A^* \leq_{C^*} S''\, A$ with $A$ well-formed
then there exists $S$, $t$, $b$, $C$, and $S'$ such that
$\mathcal{W}(A, e) = (S, t, b, C)$
$S'' \overline{\overline{\;\;}}_{NF(A,e)}\; S'\, S$
$C^*, S''\, A \mid e : \sigma^* \,\&\, b^* \preceq^{S'} C, S\, A \mid e : GEN(S\, A, b)(C, t) \,\&\, b$

**Proof** See Appendix D. □

## 5.6 Relation to ML Typing

In Sect. 2.8 we demonstrated that programs typeable in the (pure functional) ML type system can be typed in our system; we shall now elaborate on this and show that such programs are in fact also accepted by our algorithm $\mathcal{W}$.

Let $e$ be a closed sequential expression belonging to *Exp*; and let $A$ be as in Figure 2.4 but restricted to sequential constants (then $A$ is trivially well-formed). Suppose that

$$\epsilon(A) \vdash_n^{\mathrm{ML}} e : u.$$

Then Theorem 2.25 tells us that there exists $t^*$ with $\epsilon(t^*) = u$ such that

$$C_\beta, A \vdash_n e : t^* \,\&\, \beta \tag{1}$$

where $C_\beta = \{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}$. By examining the proof of Theorem 2.25, we see that we can assume that the inference (1) is atomic. Hence we can apply Theorem 5.18 (with $S'' = \mathrm{Id}$) to infer that

$$\mathcal{W}(A, e) \text{ succeeds with result } (S, t, b, C)$$

and that there exists $S'$ such that

$$C_\beta, A \mid e : t^* \,\&\, \beta \preceq^{S'} C, S\,A \mid e : \mathit{GEN}(S\,A, b)(C, t) \,\&\, b.$$

In particular, we have $t^* \leq_{C_\beta} S'\,(\mathit{GEN}(S\,A, b)(C, t))$ implying that

$$C_\beta \vdash S^*\,t \subseteq t^* \text{ holds for some } S^*$$

so $t$ is a "most general type".

# Chapter 6

# Post-processing the Inference Algorithm

In Chap. 4 we saw that our reconstruction algoritm $\mathcal{W}$ when applied successfully to a given program $e$ returns a quadruple $(S, t, b, C)$; here $S$ is of no interest (since the top-level environment contains no free variables), and $t$ will in many cases be `unit`. What we are really interested in is the behaviour $b$, and the relation between the variables occurring there as given by $C$; this constraint set is atomic (cf. Lemma 4.29) and may be quite large in spite of the size reduction performed by $\mathcal{R}$ (Sect. 4.4). In this chapter we describe how to transform the constraints, and at the same time simplify $b$, so as to improve readability.

In Sect. 6.1 we shall see how to find a solution $\mathsf{R}$ to the region constraints $C^r$. The user will typically, as illustrated in [3, 15], restrict his attention to a few selected channel labels; with $\mathsf{L}_{\text{hid}}$ the remaining "hidden" labels we introduce a special behaviour $\tau$ which denotes creation of, or communication over, a channel whose label belongs to $\mathsf{L}_{\text{hid}}$.

With $\mathsf{R}$ and $\mathsf{L}_{\text{hid}}$ given, Sect. 6.2 lists a number of basic techniques that can be used to manipulate the behaviour $b$ and the behaviour constraints $C^b$. In Sect. 6.4 we shall see that these transformations are in fact correct, using the correctness criterion[1] from Sect. 6.3 which is expressed using bisimulations

[1] We do not aim at "syntactic" correctness or even just soundness, as Theorem 4.31 (stating that $C, A \vdash_n e : t \,\&\, b$) apparently cannot be extended to state that also the result of post-processing corresponds to a valid inference.

as is well-known from other process algebras.

**Example 6.1** Suppose $\mathcal{W}$ returns the behaviour $\beta_0; \beta_1; (SPAWN\, \beta_2); \beta_3$, together with the region constraints

$$\{0\} \subseteq \rho_0, \qquad \{1\} \subseteq \rho_1, \qquad \rho_1 \subseteq \rho_2$$

and the behaviour constraints

$$\begin{array}{ll} \texttt{unit}\ \mathrm{CHAN}\ \rho_0 \subseteq \beta_0, & \texttt{unit}\ \mathrm{CHAN}\ \rho_1 \subseteq \beta_1, \\ \rho_2\,!\,\texttt{unit}; \rho_0\,?\,\texttt{unit} \subseteq \beta_2, & \rho_2\,?\,\texttt{unit}; \rho_0\,!\,\texttt{unit} \subseteq \beta_3. \end{array}$$

The mapping $\mathsf{R}$ given by $\mathsf{R}(\rho_0) = \{0\}$ and $\mathsf{R}(\rho_1) = \mathsf{R}(\rho_2) = \{1\}$ is the least solution to the region constraints, and it is possible to eliminate all behaviour constraints by "unfolding" $\beta_0, \beta_1, \beta_2, \beta_3$: in the case where $\mathsf{L}_{\mathrm{hid}} = \{1\}$ the overall behaviour is transformed into[2]

$$\texttt{unit}\ \mathrm{CHAN}\ \{0\}; \tau; SPAWN\,(\tau; \{0\}\,?\,\texttt{unit}); \tau; \{0\}\,!\,\texttt{unit}. \qquad \Box$$

**Soundness and completeness issues.** Suppose that $(b, C^b)$ has been transformed into $(b_\sim, C_\sim)$, modulo a solution $\mathsf{R}$ to $C^r$. This is still semantically sound in that "well-typed programs communicate according to their behaviour": Theorem 3.28 in essence says that the CML program is simulated by $(b, C^b)$, and in Sect. 6.4 we shall see that $(b, C^b)$ is simulated by $(b_\sim, C_\sim)$. This suggests that we can compose the results, as is formally done in Sect. 6.5.

Concerning completeness, we from Theorem 5.18 know that $b$ is a "small" (and general) behaviour, in that for any other typing involving $C^*$ and $b^*$ there exists $S'$ such that $C^* \vdash S'\, b \subseteq b^*$. In Sect. 6.4 we shall see that $(b, C^b)$ simulates $(b_\sim, C_\sim)$, indicating that also $b_\sim$ is a "small" behaviour. In Sect. 6.1 we shall argue that $\mathsf{R}$ is in some sense "principal" and hence we might be tempted to say that the algorithm $\mathcal{W}$, augmented with post-processing, is complete wrt. the inference system; it seems hard, however, to formalise this claim in a meaningful way, and hence we shall refrain from such an attempt.

[2]When printing behaviours, we often replace $\rho$ by the value of $\mathsf{R}(\rho) \setminus \mathsf{L}_{\mathrm{hid}}$.

## 6.1 Solving Region Constraints

Let $R$ be a mapping from region variables into subsets of some universe (which includes $Lab$); we say that $R$ is a solution to the region constraints $C$, to be written $R \models C$, if for all $(r_1 \subseteq r_2) \in C$ it holds that $R(r_1) \subseteq R(r_2)$. (Here $R(\{l\}) = \{l\}$.)

**Fact 6.2** Suppose that $R \models C^r$, with $C$ atomic. If $C \vdash r_1 \subseteq r_2$ then $R(r_1) \subseteq R(r_2)$.

**Proof** As $C$ is consistent (Fact 4.2), Corollary 2.28 tells us that $\overline{C} \vdash_{fw} (r_1 \subseteq r_2)$. The claim now follows from a trivial induction in this derivation, employing that $(\overline{C})^r = C^r$. □

The region constraints returned by $\mathcal{W}$ are of the form $\rho' \subseteq \rho$ or $\{l\} \subseteq \rho$ (the former kind may be produced when $\mathcal{F}$ decomposes a type and the latter when analysing $\texttt{channel}^l$). Clearly there exists a least solution to these constraints, mapping each region variable into a set of labels, and it is computable using standard iteration techniques.

The least solution, however, is not necessarily the one of interest, as demonstrated by the program

$$\texttt{rec}\ f\ ch \Rightarrow \texttt{if}\ \ldots\ \texttt{then}\ ch\ \texttt{else}\ \texttt{channel}^0\ \texttt{()}$$

for which $\mathcal{W}$ will infer the type

$$\alpha\ \texttt{chan}\ \rho\ \rightarrow^{\beta}\ \alpha\ \texttt{chan}\ \rho$$

and also generate the constraint $\{0\} \subseteq \rho$. The value returned by the program may be a channel allocated by $\texttt{channel}^0$ but it may also be a channel given as input, and the latter possibility is not recorded by the least solution which maps $\rho$ to $\{0\}$; therefore we shall, in order to obtain some "principality", rather prefer a solution which maps $\rho$ to $\{0\} \cup \{\rho\}$.

The above can be generalised, observing that "input channels" apparently correspond to region variables occurring negatively in the overall type: a solution should map this kind of variable into a set containing not only labels but also a meta variable (the variable itself can be used). Again it is clearly possible to compute the least such solution, to be denoted $\mathsf{R}$.

## 6.2 A Catalogue of Behaviour Transformations

In this section we list a selection of basic transformation steps (assuming a fixed mapping $\mathsf{R}$ and a set of hidden labels $\mathsf{L}_{\text{hid}}$), operating on *process configurations*: pairs of the form $(b, C)$ where $C$ contains behaviour constraints only. In Sect. 6.4 we shall see that if $(b, C)$ in a number of such steps is transformed into $(b', C')$, then $(b, C)$ and $(b', C')$ are bisimilar (modulo $\mathsf{R}$ and $\mathsf{L}_{\text{hid}}$), as defined in Sect. 6.3. The catalogue is not exhaustive, and the inclusion of other techniques may be beneficial to further enhance readability.

**Auxiliary notions.** A behaviour is a *channel action* if it takes the form $t$ CHAN $\rho$ or $\rho!t$ or $\rho?t$; the region part $\rho$ of a channel action $ca$ is denoted $ca^r$.

Most transformation steps can be expressed as *homomorphisms*:

**Definition 6.3** Let $F$ map channel actions into channel actions or $\tau$, and map behaviour variables into arbitrary behaviours. The homomorphism induced by $F$, to be denoted $\mathcal{S}_F$, is the mapping from behaviours into behaviours given by

$$
\begin{array}{lcl}
\mathcal{S}_F(\beta) & = & F(\beta) \\
\mathcal{S}_F(\varepsilon) & = & \varepsilon \\
\mathcal{S}_F(b_1; b_2) & = & \mathcal{S}_F(b_1); \mathcal{S}_F(b_2) \\
\mathcal{S}_F(b_1 + b_2) & = & \mathcal{S}_F(b_1) + \mathcal{S}_F(b_2) \\
\mathcal{S}_F(\mathit{SPAWN}\, b) & = & \mathit{SPAWN}\, \mathcal{S}_F(b) \\
\mathcal{S}_F(ca) & = & F(ca) \\
\mathcal{S}_F(\tau) & = & \tau
\end{array}
$$

$\mathcal{S}_F$ can in the obvious way be extended to operate on behaviour constraints:

$$\mathcal{S}_F(C) = \{(\mathcal{S}_F(b_1) \subseteq \mathcal{S}_F(b_2)) \mid (b_1 \subseteq b_2) \in C\}.$$

### 6.2.1 Simplification

A behaviour or a constraint set may be simplified into something equivalent: $(b, C)$ can be transformed into $(b', C')$, provided

$$C \vdash C' \text{ and } C' \vdash C \text{ and } C \vdash b \equiv b'.$$

A very frequent application is to replace $b; \varepsilon$ or $\varepsilon; b$ by $b$.

### 6.2.2 Hiding

Channel actions, not affecting the channels of interest, may be replaced by $\tau$ (this step needs to be done only once): if $C_0$ is well-formed then $(b_0, C_0)$ can be transformed into $(\mathcal{H}(b_0), \mathcal{H}(C_0))$, where $\mathcal{H}$ is the homomorphism induced by $F_h$ given below.

$F_h(ca) = \tau$ provided $\mathsf{R}(\rho) \subseteq \mathsf{L}_{\text{hid}}$, where $\rho = ca^r$;
otherwise $F_h$ behaves as the identity.

### 6.2.3 Unfolding

Suppose that the well-formed constraint set $C_0$ contains one and only one constraint with $\beta_0$ on the right hand side, namely $(b'_0 \subseteq \beta_0)$; and further suppose that *(i)* $\beta_0$ does not occur in $b'_0$, and *(ii)* $\beta_0$ does not[3] belong to *ChanVar*$(b_0, C_0)$ (cf. Sect. 4.4). Then $\beta_0$ may be unfolded into $b'_0$, that is $(b_0, C_0)$ can be transformed into $(\mathcal{U}(b_0), \mathcal{U}(C_0))$, where $\mathcal{U}$ is the homomorphism induced by $F_u$ given below.

$F_u(\beta_0) = b'_0$ and otherwise $F_u$ behaves as the identity.

Simplification (cf. Sect. 6.2.1) often occurs in connection with unfolding:

- To prepare for unfolding, it may be necessary to replace two constraints $\{b_1 \subseteq \beta_0, b_2 \subseteq \beta_0\}$ by a single constraint $(b_1 + b_2 \subseteq \beta_0)$.

[3]This requirement is needed, since a behaviour variable in *ChanVar*() occurs inside some type and hence cannot be replaced by a non-variable behaviour.

- After unfolding, $\mathcal{U}(C_0)$ is not necessarily well-formed as it contains the constraint $\mathcal{U}(b_0') \subseteq b_0'$; but due to requirement *(i)* above this is the identity and hence it can be eliminated.

To prevent "code explosion", unfolding should be performed only if either *(i)* there is at most one occurrence of $\beta_0$ in $b_0$ and the left hand sides of $C_0$, or *(ii)* $b_0'$ is very small (for example $\varepsilon$).

### 6.2.4 Collapsing

Let $C_0$ be well-formed, and suppose that $\beta_0'$ and $\beta_0''$ are in some sense (to be specified soon) "equivalent" wrt. $C_0$; then $\beta_0'$ may be collapsed into $\beta_0''$: $(b_0, C_0)$ can be transformed into $(\mathcal{C}(b_0), \mathcal{C}(C_0))$, where $\mathcal{C}$ is the homomorphism induced by $F_c$ given below.

$F_c$ replaces all occurrences of $\beta_0'$ with $\beta_0''$.

Below we list two conditions, each of which is sufficient for this step to be valid:

**Cycles:** $C_0 \vdash \beta_0' \equiv \beta_0''$ holds.[4].

**Sharing code:** the only constraints in $C_0$ with $\beta_0'$ or $\beta_0''$ on the right hand sides are $(b_0' \subseteq \beta_0')$ and $(b_0'' \subseteq \beta_0'')$, where $b_0'' = \mathcal{C}(b_0')$; notice that these constraints will give rise to one constraint only in $\mathcal{C}(C_0)$. (Example: $C_0$ contains the constraints $\rho\,!\,\mathtt{int}; \beta_0' \subseteq \beta_0'$ and $\rho\,!\,\mathtt{int}; \beta_0'' \subseteq \beta_0''$.)

## 6.3 The Notion of Bisimulation

In this section we formally define the notion of (strong) bisimulation; the intention is that two process configurations are bisimilar if any sequence of "actions" performed by the first can be "simulated" by the second, and vice

[4]Notice that also cycles where two or more elements belong to $\mathit{ChanVar}(b_0, C_0)$ may be collapsed, something $\mathcal{R}$ does not allow; the reason why we can be more liberal here is that we consider a correctness criterion based on the notion of bisimulation, rather than on the inference system from Fig. 2.5.

$$
\begin{array}{l}
(b_1, C_1) \sim (b_2, C_2) \\
\qquad \text{if} \quad C_1 \vdash b_1 \rightarrow^{a_1} b_1' \Rightarrow \exists a_2, b_2': \\
\qquad\qquad C_2 \vdash b_2 \rightarrow^{a_2} b_2' \wedge (a_1, C_1) \dot{\sim} (a_2, C_2) \wedge (b_1', C_1) \sim (b_2', C_2) \\
\qquad \text{and} \; C_2 \vdash b_2 \rightarrow^{a_2} b_2' \Rightarrow \exists a_1, b_1': \\
\qquad\qquad C_1 \vdash b_1 \rightarrow^{a_1} b_1' \wedge (a_1, C_1) \dot{\sim} (a_2, C_2) \wedge (b_1', C_1) \sim (b_2', C_2)
\end{array}
$$

Figure 6.1: The bisimulation relation $\sim$

versa. In our setting, an *action* $a$ is a behaviour which is either a channel action $ca$ or of the form $\mathit{SPAWN}\, b$ (a spawn action) or $\tau$ (a hidden action); notice that a homomorphism $\mathcal{S}_F$ maps actions into actions. The inference system from Fig. 2.7 gives rise to a transition relation, labelled with actions, on process configurations; the intuition is that if $C \vdash a; b' \subseteq b$ then one of the "options" of $b$ is to first perform $a$ and then become $b'$.

**Definition 6.4** We write $C \vdash b \rightarrow^a b'$ if $C \vdash a; b' \subseteq b$. □

We shall introduce a relation $\sim$ on process configurations and another relation $\dot{\sim}$ on *action configurations*, i.e. pairs $(a, C)$ with $a$ an action and $C$ a set of behaviour constraints; these relations are implicitly parametrised with respect to the given $\mathsf{R}$ and $\mathsf{L}_{\text{hid}}$. Our aim is that $\sim$ and $\dot{\sim}$ should enjoy the properties stated in Figs. 6.1 and 6.2, expressing their mutual dependency; these properties seem fairly natural. (Example: $(\rho!(\texttt{int} \rightarrow^{\beta_1} \texttt{int}), C_1) \dot{\sim} (\rho!(\texttt{int} \rightarrow^{\beta_2} \texttt{int}), C_2)$ will hold provided $(\beta_1, C_1) \sim (\beta_2, C_2)$.)

We can in fact view Figs. 6.1 and 6.2 as *definitions*: since their right hand sides give rise to a monotone functional $\mathcal{G}$ on the complete lattice of relations[5], ordered by subset inclusion, we can stipulate $\sim \cup \dot{\sim}$ to be its *greatest* fixed point (guaranteed to exist by Tarski's theorem).

The following proof principle is most useful for reasoning about $\sim$ and $\dot{\sim}$:

**Observation 6.5** Suppose we want to check that for some relation $Q$ it holds that

[5] The elements in this lattice are the subsets of $PC \times PC \cup AC \times AC$, where $PC$ is the set of process configurations and $AC$ is the set of action configurations, and where with some misuse of notation we write $\cup$ for "disjoint union"; therefore each lattice element $Q$ may be uniquely written as $Q_p \cup Q_a$ where $Q_p$ is a subset of $PC \times PC$ and $Q_a$ is a subset of $AC \times AC$.

$(SPAWN\, b_1, C_1) \dot{\sim} (SPAWN\, b_2, C_2)$
if $(b_1, C_1) \sim (b_2, C_2)$

$(\tau, C_1) \dot{\sim} (\tau, C_2)$

$(ca, C_1) \dot{\sim} (\tau, C_2)$
if $\mathsf{R}(\rho) \subseteq \mathsf{L}_{\mathrm{hid}}$, where $\rho = ca^r$

$(ca, C_1) \dot{\sim} (\phi\,(ca), C_2)$
if $\phi$ is a substitution with only behaviour variables in $Dom(\phi)$
and $\forall \beta \in FV(ca) : (\beta, C_1) \sim (\phi\,\beta, C_2)$

$(a_1, C_1) \dot{\sim} (a_2, C_2)$
if $(a_2, C_2) \dot{\sim} (a_1, C_1)$

$(a_1, C_1) \dot{\sim} (a_2, C_2)$
if $\exists (a_3, C_3) : (a_1, C_1) \dot{\sim} (a_3, C_3) \wedge (a_3, C_3) \dot{\sim} (a_2, C_2)$

Figure 6.2: The relation $\dot{\sim}$ on action configurations

$$Q \subseteq (\sim \cup \dot{\sim}). \tag{1}$$

Then it is sufficient to show

$$Q \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim}). \tag{2}$$

For as $(\sim \cup \dot{\sim}) = \mathcal{G}(\sim \cup \dot{\sim}) \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim})$ holds by monotonicity of $\mathcal{G}$, (2) ensures that $(Q \cup \sim \cup \dot{\sim}) \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim})$ which (again employing Tarski's theorem) is enough to establish $(Q \cup \sim \cup \dot{\sim}) \subseteq (\sim \cup \dot{\sim})$ and hence (1). □

As to be expected, $\sim$ and $\dot{\sim}$ are equivalence relations:

**Fact 6.6** The relations $\sim$ and $\dot{\sim}$ are reflexive, symmetric, and transitive.

**Proof** See Appendix E. □

## 6.4 Correctness of the Transformations

In Sect. 6.2 we have listed a number of techniques for transforming one process configuration $(b, C)$ into another $(b', C')$; we shall now demonstrate that all these techniques are "correct" in the sense that $(b, C) \sim (b', C')$. As $\sim$ is reflexive and transitive (Fact 6.6), this shows that if $(b_0, C_0)$ is transformed into $(b_n, C_n)$ via a sequence of such steps, then $(b_0, C_0) \sim (b_n, C_n)$.

Before examining the techniques in turn, we establish some general results about homomorphisms :

**Lemma 6.7** Let $C$ be a set of behaviour constraints, and let $\mathcal{S}_F$ be a homomorphism. If $C \vdash b_1 \subseteq b_2$ then also

1. $\mathcal{S}_F(C) \vdash \mathcal{S}_F(b_1) \subseteq \mathcal{S}_F(b_2)$
2. $\mathit{ChanVar}(b_1) \subseteq \mathit{ChanVar}(b_2, C)$.

**Proof** As $C$ is consistent (Fact 5.11), Corollary 2.28 tells us that $C \vdash_{fw} b_1 \subseteq b_2$; the claims now follow from a straightforward induction in this derivation, making use of the homomorphism properties. □

Applying the lemma on judgements of the form $C \vdash a; b_1 \subseteq b$ yields

**Corollary 6.8** Let $C$ be a set of behaviour constraints, and let $\mathcal{S}_F$ be a homomorphism. If $C \vdash b \rightarrow^a b_1$ then

1. $\mathcal{S}_F(C) \vdash \mathcal{S}_F(b) \rightarrow^{\mathcal{S}_F(a)} \mathcal{S}_F(b_1)$
2. $\mathit{ChanVar}(a, b_1) \subseteq \mathit{ChanVar}(b, C)$.

**Lemma 6.9** Let $C$ be a set of behaviour constraints, and let $\mathcal{S}_F$ be a homomorphism with the following properties:

1. if for some $b'_1$ and $b_2$ it holds that $(b'_1 \subseteq \mathcal{S}_F(b_2)) \in \mathcal{S}_F(C)$, then there exists $b_1$ with $\mathcal{S}_F(b_1) = b'_1$ such that $C \vdash b_1 \subseteq b_2$;
2. if for some $\beta$ it holds that $F(\beta)$ is not a variable, then

$$C \vdash F(\beta) \subseteq \beta \text{ and } \mathcal{S}_F(F(\beta)) = F(\beta).$$

We then have the following implications:

1. if $\mathcal{S}_F(C) \vdash b'_1 \subseteq \mathcal{S}_F(b_2)$ there exists $b_1$ with $\mathcal{S}_F(b_1) = b'_1$ such that $C \vdash b_1 \subseteq b_2$;

2. if $\mathcal{S}_F(C) \vdash \mathcal{S}_F(b_2) \rightarrow^{a'} b'_0$ there exists $a$, $b_0$ with $\mathcal{S}_F(a) = a'$ and $\mathcal{S}_F(b_0) = b'_0$ such that $C \vdash b_2 \rightarrow^{a} b_0$.

**Proof** See Appendix E. □

### 6.4.1 Simplification

We define a relation $Q$ on process configurations:

$$(b_1, C_1)\; Q\; (b_2, C_2) \text{ if } C_1 \vdash C_2 \text{ and } C_2 \vdash C_1 \text{ and } C_1 \vdash b_1 \equiv b_2$$

and the correctness of simplification (Sect. 6.2.1) can be demonstrated by proving $Q \subseteq\sim \cup \dot{\sim}$; by Observation 6.5 it is sufficient to establish

$$Q \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim}).$$

So consider $(b_1, C_1)\; Q\; (b_2, C_2)$. First assume $C_1 \vdash b_1 \rightarrow^{a_1} b'_1$, that is $C_1 \vdash a_1; b'_1 \subseteq b_1 \equiv b_2$, so by Lemma 2.19 we also have $C_2 \vdash a_1; b'_1 \subseteq b_2$, that is $C_2 \vdash b_2 \rightarrow^{a_1} b'_1$.

Next assume $C_2 \vdash b_2 \rightarrow^{a_2} b'_2$, that is $C_2 \vdash a_2; b'_2 \subseteq b_2$, so by Lemma 2.19 we also have $C_1 \vdash a_2; b'_2 \subseteq b_2 \equiv b_1$, that is $C_1 \vdash b_1 \rightarrow^{a_2} b'_2$.

As $\sim$ and $\dot{\sim}$ are reflexive (Fact 6.6), this shows the desired relation

$$(b_1, C_1)\; \mathcal{G}(Q \cup \sim \cup \dot{\sim})\; (b_2, C_2).$$

### 6.4.2 Hiding

In order to show the correctness of transforming $(b_0, C_0)$ into $(\mathcal{H}(b_0), \mathcal{H}(C_0))$, cf. Sect. 6.2.2, we define a relation $Q$ on process configurations and action configurations by stipulating

$$\forall b : (b, C_0) \; Q \; (\mathcal{H}(b), \mathcal{H}(C_0))$$
$$\forall a : (a, C_0) \; Q \; (\mathcal{H}(a), \mathcal{H}(C_0))$$

Then correctness can be demonstrated by proving $Q \subseteq (\sim \cup \dot{\sim})$; by Observation 6.5 it is sufficient to establish $Q \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim})$.

First consider $(a, C_0) \; Q \; (\mathcal{H}(a), \mathcal{H}(C_0))$, our aim is to show

$$(a, C_0) \; \mathcal{G}(Q \cup \sim \cup \dot{\sim}) \; (\mathcal{H}(a), \mathcal{H}(C_0)). \tag{3}$$

If $a$ is a channel action $ca$ with $\rho = ca^r$, we distinguish between two cases:

- if $\mathsf{R}(\rho) \subseteq \mathsf{L}_{\mathrm{hid}}$ then $\mathcal{H}(a) = \tau$, so clearly (3) holds;
- if $\mathsf{R}(\rho) \setminus \mathsf{L}_{\mathrm{hid}} \neq \emptyset$ then $\mathcal{H}(a) = a$, to establish (3) observe that for all $\beta$ we have $\mathcal{H}(\beta) = \beta$ and hence $(\beta, C_0) \; Q \; (\beta, \mathcal{H}(C_0))$.

If $a$ is a spawn action $\mathit{SPAWN}\, b$ then $\mathcal{H}(a) = \mathit{SPAWN}\, \mathcal{H}(b)$ from which we infer (3).

The remaining possibility is that $a$ is a hidden action $\tau$, then $\mathcal{H}(a) = \tau$ and (3) trivially holds.

Next consider $(b, C_0) \; Q \; (\mathcal{H}(b), \mathcal{H}(C_0))$, our aim is to show

$$(b, C_0) \; \mathcal{G}(Q \cup \sim \cup \dot{\sim}) \; (\mathcal{H}(b), \mathcal{H}(C_0)). \tag{4}$$

By Corollary 6.8 it holds that

$$C_0 \vdash b \to^a b_1 \text{ implies } \mathcal{H}(C_0) \vdash \mathcal{H}(b) \to^{\mathcal{H}(a)} \mathcal{H}(b_1)$$

which provides the "one half" of (4); for the "other half" assume that

$$\mathcal{H}(C_0) \vdash \mathcal{H}(b) \to^{a'} b'_1$$

and we would like to find $a$ and $b_1$ with $\mathcal{H}(a) = a'$ and $\mathcal{H}(b_1) = b'_1$ such that

$$C_0 \vdash b \to^a b_1.$$

Lemma 6.9 will provide these $a$ and $b_1$ so we must check that the conditions for applying this lemma are fulfilled: Condition 2 is vacuously true so we only need to consider Condition 1 as done below.

Let $(b_1' \subseteq \mathcal{H}(b_2))$ belong to $\mathcal{H}(C_0)$, that is (as $C_0$ is well-formed) there exists $\beta$ and $b_1$ such that $(b_1 \subseteq \beta) \in C_0$ and $\mathcal{H}(\beta) = \mathcal{H}(b_2)$ and $\mathcal{H}(b_1) = b_1'$. As $\mathcal{H}(\beta) = \beta$ we deduce that $b_2 = \beta$, hence we have the desired judgement $C_0 \vdash b_1 \subseteq b_2$.

### 6.4.3 Unfolding

To show the correctness of transforming $(b_0, C_0)$ into $(\mathcal{U}(b_0), \mathcal{U}(C_0))$, cf. Sect. 6.2.3, we define a relation $Q$ on process configurations and action configurations by stipulating

$$\begin{array}{ll} \forall b \text{ with } \beta_0 \notin \mathit{ChanVar}(b): & (b, C_0)\ Q\ (\mathcal{U}(b), \mathcal{U}(C_0)) \\ \forall a \text{ with } \beta_0 \notin \mathit{ChanVar}(a): & (a, C_0)\ Q\ (\mathcal{U}(a), \mathcal{U}(C_0)) \end{array}$$

Then correctness can be demonstrated by proving $Q \subseteq (\sim \cup \dot{\sim})$; by Observation 6.5 it is sufficient to establish $Q \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim})$.

First consider $(a, C_0)\ Q\ (\mathcal{U}(a), \mathcal{U}(C_0))$, our aim is to show

$$(a, C_0)\ \mathcal{G}(Q \cup \sim \cup \dot{\sim})\ (\mathcal{U}(a), \mathcal{U}(C_0)). \tag{5}$$

If $a$ is a channel action then $\mathcal{U}(a) = a$, to establish (5) observe that for all $\beta \in FV(a)$ we have (as then $\beta \in \mathit{ChanVar}(a)$) $\beta \neq \beta_0$, implying $\mathcal{U}(\beta) = \beta$ and hence $(\beta, C_0)\ Q\ (\beta, \mathcal{U}(C_0))$.

If $a$ is a spawn action $\mathit{SPAWN}\, b$ then $\mathcal{U}(a) = \mathit{SPAWN}\, \mathcal{U}(b)$ from which we infer (5) since $\beta_0 \notin \mathit{ChanVar}(b)$ and hence $(b, C_0)\ Q\ (\mathcal{U}(b), \mathcal{U}(C_0))$. If $a$ is a hidden action $\tau$ then $\mathcal{U}(a) = \tau$ and (5) trivially holds.

Next consider $(b, C_0)\ Q\ (\mathcal{U}(b), \mathcal{U}(C_0))$, our aim is to show

$$(b, C_0)\ \mathcal{G}(Q \cup \sim \cup \dot{\sim})\ (\mathcal{U}(b), \mathcal{U}(C_0)). \tag{6}$$

By Corollary 6.8 it holds that (as $\beta_0 \notin \mathit{ChanVar}(C_0)$)

$$C_0 \vdash b \rightarrow^a b_1 \quad \text{implies } \mathcal{U}(C_0) \vdash \mathcal{U}(b) \rightarrow^{\mathcal{U}(a)} \mathcal{U}(b_1)$$
$$\text{with } \beta_0 \notin \mathit{ChanVar}(a, b_1)$$

which provides the "one half" of (6); for the "other half" assume that

$$\mathcal{U}(C_0) \vdash \mathcal{U}(b) \rightarrow^{a'} b_1'$$

and we would like to find $a$ and $b_1$ with $\mathcal{U}(a) = a'$ and $\mathcal{U}(b_1) = b_1'$ such that $C_0 \vdash b \rightarrow^a b_1$ (Corollary 6.8 will then ensure $\beta_0 \notin \mathit{ChanVar}(a, b_1)$, hence $(a, C_0)\ Q\ (a', \mathcal{U}(C_0))$ and $(b_1, C_0)\ Q\ (b_1', \mathcal{U}(C_0))$).

Lemma 6.9 will provide these $a$ and $b_1$ but we must check that the conditions for applying this lemma are fulfilled: concerning Condition 2, our task can be accomplished by showing $C_0 \vdash b_0' \subseteq \beta_0$ and $\mathcal{U}(b_0') = b_0'$, but this follows directly from the side conditions for unfolding.

We are left with validating Condition 1: let $b_1' \subseteq \mathcal{U}(b_2)$ belong to $\mathcal{U}(C_0)$, that is (as $C_0$ is well-formed) there exists $\beta$ and $b_1$ such that $(b_1 \subseteq \beta) \in C_0$ and $\mathcal{U}(\beta) = \mathcal{U}(b_2)$ and $\mathcal{U}(b_1) = b_1'$, we must show $C_0 \vdash b_1 \subseteq b_2$.

- if $\beta \neq \beta_0$ then $\mathcal{U}(b_2) = \mathcal{U}(\beta) = \beta$ and we deduce that $b_2$ is a variable.

  *(i)* If $b_2 = \beta_0$ then $C_0 \vdash b_1 \subseteq \beta = \mathcal{U}(b_2) = b_0' \subseteq \beta_0 = b_2$.

  *(ii)* If $b_2 \neq \beta_0$ then $b_2 = \beta$ so $C_0 \vdash b_1 \subseteq \beta = b_2$.

- if $\beta = \beta_0$, the uniqueness assumption on $b_0'$ ensures $b_1 = b_0'$. We have

  $$\mathcal{U}(b_2) = b_0'$$

  and as $\beta_0 \notin \mathit{ChanVar}(b_0')$ we therefore deduce that $\beta_0 \notin \mathit{ChanVar}(b_2)$; this shows (since $\mathcal{U}(\beta_0) = b_0'$) that if $\beta_0 \in FV(b_2)$ then $b_2 = \beta_0$.

  *(i)* If $b_2 = \beta_0$ then $C_0 \vdash b_1 = b_0' \subseteq \beta_0 = b_2$.

  *(ii)* If $\beta_0 \notin FV(b_2)$ then $C_0 \vdash b_1 = b_0' = \mathcal{U}(b_2) = b_2$.

### 6.4.4 Collapsing

To show the correctness of transforming $(b_0, C_0)$ into $(\mathcal{C}(b_0), \mathcal{C}(C_0))$, cf. Sect. 6.2.4, we define a relation $Q$ on process configurations and action configurations by stipulating

$$\forall b : (b, C_0) \; Q \; (\mathcal{C}(b), \mathcal{C}(C_0))$$
$$\forall a : (a, C_0) \; Q \; (\mathcal{C}(a), \mathcal{C}(C_0))$$

Then correctness can be demonstrated by proving $Q \subseteq (\sim \cup \dot{\sim})$; by Observation 6.5 it is sufficient to establish $Q \subseteq \mathcal{G}(Q \cup \sim \cup \dot{\sim})$.

First consider $(a, C_0) \; Q \; (\mathcal{C}(a), \mathcal{C}(C_0))$, our aim is to show

$$(a, C_0) \; \mathcal{G}(Q \cup \sim \cup \dot{\sim}) \; (\mathcal{C}(a), \mathcal{C}(C_0)). \tag{7}$$

If $a$ is a channel action then observe that $\mathcal{C}$ is a substitution with only behaviour variables in the domain, hence (7) can be established as we for all $\beta$ have $(\beta, C_0) \; Q \; (\mathcal{C}(\beta), \mathcal{C}(C_0))$.

If $a$ is a spawn action *SPAWN* $b$ then $\mathcal{C}(a) = \mathit{SPAWN}\, \mathcal{C}(b)$ from which we infer (7). If $a$ is a hidden action $\tau$ then $\mathcal{C}(a) = \tau$ and (7) trivially holds.

Next consider $(b, C_0) \; Q \; (\mathcal{C}(b), \mathcal{C}(C_0))$, our aim is to show

$$(b, C_0) \; \mathcal{G}(Q \cup \sim \cup \dot{\sim}) \; (\mathcal{C}(b), \mathcal{C}(C_0)). \tag{8}$$

By Corollary 6.8 it holds that

$$C_0 \vdash b \rightarrow^{a} b_1 \text{ implies } \mathcal{C}(C_0) \vdash \mathcal{C}(b) \rightarrow^{\mathcal{C}(a)} \mathcal{C}(b_1)$$

which provides the "one half" of (8); for the "other half" assume that

$$\mathcal{C}(C_0) \vdash \mathcal{C}(b) \rightarrow^{a'} b'_1$$

and we would like to find $a$ and $b_1$ with $\mathcal{C}(a) = a'$ and $\mathcal{C}(b_1) = b'_1$ such that

$$C_0 \vdash b \rightarrow^{a} b_1.$$

Lemma 6.9 will provide these $a$ and $b_1$ so we must check that the conditions for applying this lemma are fulfilled: Condition 2 is vacuously true, so we only need to consider Condition 1 as done below.

Let $b'_1 \subseteq \mathcal{C}(b_2)$ belong to $\mathcal{C}(C_0)$, that is (as $C_0$ is well-formed) there exists $\beta$ and $b''_1$ such that

$$(b_1'' \subseteq \beta) \in C_0 \text{ and } \mathcal{C}(\beta) = \mathcal{C}(b_2) \text{ and } \mathcal{C}(b_1'') = b_1'.$$

If $b_2 = \beta$ we define $b_1 = b_1''$ and obtain the desired relations: $\mathcal{C}(b_1) = b_1'$ and $C_0 \vdash b_1 \subseteq \beta = b_2$.

If $b_2 \neq \beta$ we infer that $\{b_2, \beta\} = \{\beta_0', \beta_0''\}$. In the case of **Cycles**, that is $C_0 \vdash \beta_0' \equiv \beta_0''$, we define $b_1 = b_1''$ and obtain the desired relations $\mathcal{C}(b_1) = b_1'$ and $C_0 \vdash b_1 \subseteq \beta \equiv b_2$. In the case of **Sharing code**, we consider two cases (we exploit that $\mathcal{C}$ is idempotent and that $b_0'' = \mathcal{C}(b_0')$):

- $\beta = \beta_0'$ and $b_2 = \beta_0''$: then (by uniqueness of $b_0'$) $b_1'' = b_0'$ and we define $b_1 = \mathcal{C}(b_1'')$, yielding $\mathcal{C}(b_1) = \mathcal{C}(\mathcal{C}(b_1'')) = \mathcal{C}(b_1'') = b_1'$ and also
$$C_0 \vdash b_1 = \mathcal{C}(b_1'') = \mathcal{C}(b_0') = b_0'' \subseteq \beta_0'' = b_2.$$
- $\beta = \beta_0''$ and $b_2 = \beta_0'$: then (by uniqueness of $b_0''$) $b_1'' = b_0''$ and we define $b_1 = b_0'$, yielding $\mathcal{C}(b_1) = \mathcal{C}(\mathcal{C}(b_0')) = \mathcal{C}(b_0'') = \mathcal{C}(b_1'') = b_1'$ and also
$$C_0 \vdash b_1 = b_0' \subseteq \beta_0' = b_2.$$

## 6.5 Semantic Soundness

So far in this chapter we have exhibited various techniques for manipulating the output from $\mathcal{W}$; we shall now demonstrate that the resulting behaviour, together with the resulting constraints, still "simulates" the CML program in question.

To accomplish this goal we reformulate and extend Theorem 3.28; here $(PB, C) \sim (PB_\sim, C_\sim)$ means that $(PB(p), C) \sim (PB_\sim(p), C_\sim)$ for all $p \in Dom(PB) = Dom(PB_\sim)$.

**Theorem 6.10** *Semantic soundness, revisited*

Let $C = C^t \cup C^b \cup C^r$ be atomic, let $\mathsf{R}$ be such that $\mathsf{R} \models C^r$ (cf. Sect. 6.1), let $\mathsf{L}_{\mathsf{hid}}$ be a set of hidden labels, and let $\sim$ and $\dot{\sim}$ be the bisimulation relations implicitly parametrised by $\mathsf{R}$ and $\mathsf{L}_{\mathsf{hid}}$ (cf. Sect. 6.3).

Let $A$ be a standard channel environment, and suppose

$$C, A \vdash_n PP : PT \,\&\, PB \text{ and } (PB, C^b) \sim (PB_\sim, C_\sim).$$

If $PP \overset{sa}{\Longleftrightarrow} PP'$ then there exists $PT'$, $PB'$, $PB'_\sim$ and a standard channel environment $A'$ such that

$$C, A' \vdash_n PP' : PT' \,\&\, PB' \text{ and } (PB', C^b) \sim (PB'_\sim, C_\sim)$$

and such that if $ch$ occurs in $PP$ then $A'(ch) = A(ch)$ and such that if $p$ is in the domain of $PP$ then *(i)* $PT'(p) = PT(p)$ and *(ii)* if $p$ is not mentioned in $sa$ then $PB'_\sim(p) = PB_\sim(p)$.

Furthermore we have the following properties:

- If $sa = p_0 \; \mathtt{chan}^l \; ch$ then there exists $t_0$ and $\rho_0$ with $l \in \mathsf{R}(\rho_0)$ such that

  $$A'(ch) = t_0 \; \mathtt{chan} \; \rho_0$$

  and there also exists action $a$ with

  $$(t_0 \; \text{CHAN} \; \rho_0, C^b) \;\dot{\sim}\; (a, C_\sim)$$

  such that

  $$C_\sim \vdash PB_\sim(p_0) \to^a PB'_\sim(p_0).$$

- If $sa = p_0 \; \mathtt{spawn} \; p'$ then there exists $a$ with

  $$(\mathit{SPAWN}\, PB'(p'), C^b) \;\dot{\sim}\; (a, C_\sim)$$

  such that

  $$C_\sim \vdash PB_\sim(p_0) \to^a PB'_\sim(p_0).$$

- If $sa = p_1, p_2 \; \mathtt{comm} \; ch$ then, with $A(ch) = t \; \mathtt{chan} \; \rho$, there exists

  $t_s$ and $t_r$ with $C \vdash t_s \subseteq t \subseteq t_r$ and

  $\rho_s$ and $\rho_r$ with $\mathsf{R}(\rho) \subseteq \mathsf{R}(\rho_s)$ and $\mathsf{R}(\rho) \subseteq \mathsf{R}(\rho_r)$

  and there exists actions $a_1$, $a_2$ with

  $(\rho_s \,!\, t_s, C^b) \;\dot{\sim}\; (a_1, C_\sim)$ and

  $(\rho_r \,?\, t_r, C^b) \;\dot{\sim}\; (a_2, C_\sim)$

  such that

$$C_\sim \vdash PB_\sim(p_1) \rightarrow^{a_1} PB'_\sim(p_1) \text{ and}$$

$$C_\sim \vdash PB_\sim(p_2) \rightarrow^{a_2} PB'_\sim(p_2).$$

**Proof** First observe that $C$ is well-formed and consistent (Fact 4.2). Hence Theorem 3.28 is applicable, yielding $A'$ and $PT'$ and $PB'$ with certain specified properties to be exploited in the sequel; also notice that if $C \vdash b_1 \subseteq b_2$ for some $b_1, b_2$ then (by Corollary 2.28) we have $\overline{C} \vdash_{fw} b_1 \subseteq b_2$ which by a trivial induction (using $(\overline{C})^b = C^b$) implies $C^b \vdash b_1 \subseteq b_2$. We perform case analysis on the semantic action $sa$; in all cases we for $p$ not mentioned in $sa$ use $PB'_\sim(p) = PB_\sim(p)$, establishing $(PB'(p), C^b) \sim (PB'_\sim(p), C_\sim)$ for such $p$.

$\underline{sa = \texttt{seq}}$: The claim is trivial.

$\underline{sa = p_0 \ \texttt{chan}^l \ ch}$: We know that $C \vdash \{l\} \subseteq \rho_0$ which by Fact 6.2 implies $l \in \mathsf{R}(\rho_0)$; and we know

$$C^b \vdash t_0 \ \text{CHAN} \ \rho_0; PB'(p_0) \subseteq PB(p_0)$$

which as $(PB(p_0), C^b) \sim (PB_\sim(p_0), C_\sim)$ implies the existence of $a$ and $b'$ such that $C_\sim \vdash PB_\sim(p_0) \rightarrow^a b'$ with $(t_0 \ \text{CHAN} \ \rho_0, C^b) \dot{\sim} (a, C_\sim)$ and $(PB'(p_0), C^b) \sim (b', C_\sim)$. We can thus define $PB'_\sim(p_0) = b'$ to obtain the desired properties.

$\underline{sa = p_0 \ \texttt{spawn} \ p'}$: We know that

$$C^b \vdash (SPAWN\, PB'(p')); PB'(p_0) \subseteq PB(p_0)$$

which as $(PB(p_0), C^b) \sim (PB_\sim(p_0), C_\sim)$ implies the existence of $a$ and $b'$ such that $C_\sim \vdash PB_\sim(p_0) \rightarrow^a b'$ with $(SPAWN\, PB'(p'), C^b) \dot{\sim} (a, C_\sim)$ and $(PB'(p_0), C^b) \sim (b', C_\sim)$, we can thus define $PB'_\sim(p_0) = b'$ to obtain the desired properties.

$\underline{sa = p_1, p_2 \ \texttt{comm} \ ch}$: We know that $C \vdash \rho \subseteq \rho_s$ and $C \vdash \rho \subseteq \rho_r$ which by Fact 6.2 implies $\mathsf{R}(\rho) \subseteq \mathsf{R}(\rho_s)$ and $\mathsf{R}(\rho) \subseteq \mathsf{R}(\rho_r)$; and we know

$$C^b \vdash (\rho_s \,!\, t_s); PB'(p_1) \subseteq PB(p_1)$$

$$C^b \vdash (\rho_r \,?\, t_r); PB'(p_2) \subseteq PB(p_2)$$

which as $(PB(p_1), C^b) \sim (PB_\sim(p_1), C_\sim)$ and $(PB(p_2), C^b) \sim (PB_\sim(p_2), C_\sim)$ implies the existence of $a_1, a_2$ and $b'_1, b'_2$ such that

$$C_\sim \vdash PB_\sim(p_1) \rightarrow^{a_1} b'_1 \text{ and } C_\sim \vdash PB_\sim(p_2) \rightarrow^{a_2} b'_2$$

with $(PB'(p_1), C^b) \sim (b'_1, C_\sim)$ and $(PB'(p_2), C^b) \sim (b'_2, C_\sim)$ and with

$$(\rho_s \,!\, t_s, C^b) \mathrel{\dot\sim} (a_1, C_\sim) \text{ and } (\rho_r \,?\, t_r, C^b) \mathrel{\dot\sim} (a_2, C_\sim).$$

We can thus define $PB'_\sim(p_1) = b'_1$ and $PB'_\sim(p_2) = b'_2$ to obtain the desired properties. □

### 6.5.1 Semantic Soundness of the Overall System

Let $A$ be as in Figure 2.4, and suppose that $\mathcal{W}(A, e)$ succeeds with result $(S, t, b, C)$. As $A$ is closed and well-formed (Fact 2.16), it by Theorem 4.31 holds that

$$C, A \vdash_n e : t \,\&\, b$$

where $C$ is atomic by Lemma 4.29. Next[6] suppose that the methods from Sect. 6.1 are applied to find $\mathsf{R}$ such that $\mathsf{R} \models C^r$. Finally suppose that $(b, C^b)$ is transformed, using the methods from Sect. 6.2 and modulo this $\mathsf{R}$ and some $\mathsf{L}_{\text{hid}}$, into $(b_\sim, C_\sim)$. From Sect. 6.4 we know that

$$(b, C^b) \sim (b_\sim, C_\sim)$$

and hence we are in position to apply Theorem 6.10.

[6] Concerning the type constraints in $C$, the system may as an additional feature collapse all cycles: with $S$ a substitution unifying all $\alpha_1, \alpha_2$ with $C \vdash \alpha_1 \equiv \alpha_2$, we then have $C', A \vdash_n e : t' \,\&\, b'$ with $C' = S\,C$ and $t' = S\,t$ and $b' = S\,b$; subsequently the system operates on these entities rather than on $(t, b, C)$.

# Chapter 7

# Conclusion

We have developed a type and effect system for a core subset of CML. The effects include regions and causality information; the type system includes polymorphism (á la ML) and subtyping (induced by the ordering on effects).

The type system is proved to be sound wrt. a small-step semantics, in the sense that a subject reduction result holds. An inference algorithm is presented; it is sound and (in a certain sense) also complete.

The constraints produced by the algorithm can be post-processed so as to be quite readable and informative, this is illustrated by our prototype implementation which can be experienced at

`http://www.daimi.aau.dk/~bra8130/TBAcml/TBA_CML.html`.

The system accepts programs written in a non-trivial subset of CML and these are by the front end translated into our core subset, extended with a bunch of extra constants.

We believe our approach to be rather open-ended, in the sense that extra features can be added to the language or type system with a limited effort; similarly it seems that many of the ideas may be transfered and applied to other settings.

**Acknowledgements.** This research has been supported in part by the DART (Danish Science Research Council) and LOMAPS (ESPRIT BRA 8130) projects.

# Appendix A

# Proofs of Results Concerning the Basic Framework

## Basic properties of the inference system

**Lemma 2.18** For all substitutions $S$:

(a) If $C \vdash C_0$ then $S\,C \vdash S\,C_0$ (and has the same shape).

(b) If $C, A \vdash e : \sigma \,\&\, b$ then $S\,C, S\,A \vdash e : S\,\sigma \,\&\, S\,b$ (and has the same shape).

**Proof** To establish (a), we prove that $C \vdash g_1 \subseteq g_2$ entails $S\,C \vdash S\,g_1 \subseteq S\,g_2$ (with the same shape); this is straightforward by induction. For the claim (b) we proceed by induction on the inference.

For the cases (con) and (id) the claim is immediate, and for the cases (abs), (app), (sapp), (let), (rec), (if) it follows directly using the induction hypothesis. For the case (sub) we use (a) together with the induction hypothesis.

**The case (ins).** Then $C, A \vdash e : S_0\,t_0 \,\&\, b$ because with $C \vdash S_0\,C_0$ and $Dom(S_0) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ we have $C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$, and wlog. (cf. Observation 2.4) we can assume that $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ is disjoint from $(Dom(S) \cup Ran(S))$. The induction hypothesis gives

$$S\,C, S\,A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : S\,C_0).\ S\,t_0 \,\&\, S\,b. \qquad (1)$$

From (a) we get $S\,C \vdash S\,S_0\,C_0$. Let $S_0' = [\vec{\alpha}\vec{\beta}\vec{\rho} \mapsto S\,S_0\,(\vec{\alpha}\vec{\beta}\vec{\rho})]$, then on $FV(t_0, C_0)$ it holds that $S_0'\,S = S\,S_0$. Therefore $S\,C \vdash S_0'\,S\,C_0$, so we can apply (ins) on (1) with $S_0'$ as the "instance substitution" to get $S\,C, S\,A \vdash e : S_0'\,S\,t_0\,\&\,S\,b$. Since $S_0'\,S\,t_0 = S\,S_0\,t_0$ this is the required result.

**The case (gen).** Then $C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0\,\&\,b$ holds because

$$C \cup C_0, A \vdash e : t_0\,\&\,b$$

$$\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \text{ is well-formed} \tag{2}$$

$$\text{there exists } S_0 \text{ with } Dom(S_0) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \text{ such that } C \vdash S_0\,C_0 \tag{3}$$

$$\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset \tag{4}$$

Define $R = [\vec{\alpha}\vec{\beta}\vec{\rho} \mapsto \vec{\alpha'}\vec{\beta'}\vec{\rho'}]$ with $\{\vec{\alpha'}\vec{\beta'}\vec{\rho'}\}$ fresh. We then apply the induction hypothesis (with $S\,R$) and due to (4) this gives us

$$S\,C \cup S\,R\,C_0, S\,A \vdash e : S\,R\,t_0\,\&\,S\,b.$$

Below we prove

$$\forall(\vec{\alpha'}\vec{\beta'}\vec{\rho'} : S\,R\,C_0).\ S\,R\,t_0 = S\,(\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0) \text{ is well-formed} \tag{5}$$

$$\text{there exists } S' \text{ with } Dom(S') \subseteq \{\vec{\alpha'}\vec{\beta'}\vec{\rho'}\} \text{ such that } S\,C \vdash S'\,S\,R\,C_0 \tag{6}$$

$$\{\vec{\alpha'}\vec{\beta'}\vec{\rho'}\} \cap FV(S\,C, S\,A, S\,b) = \emptyset \tag{7}$$

It then follows that $S\,C, S\,A \vdash e : S\,(\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0)\,\&\,S\,b$ as required. Clearly the inference has the same shape.

First we observe that (5) follows from (2) and Fact 2.13. For (6) define $S' = [\vec{\alpha'}\vec{\beta'}\vec{\rho'} \mapsto S\,S_0\,(\vec{\alpha}\vec{\beta}\vec{\rho})]$. From (3) and (a) we get $S\,C \vdash S\,S_0\,C_0$. Since $S'\,S\,R = S\,S_0$ on $FV(C_0)$ the result follows. Finally (7) holds trivially by choice of $\vec{\alpha'}\vec{\beta'}\vec{\rho'}$. □

**Lemma 2.19** For all sets $C'$ of constraints satisfying $C' \vdash C$:

(a) If $C \vdash C_0$ then $C' \vdash C_0$.

(b) If $C, A \vdash e : \sigma\,\&\,b$ then $C', A \vdash e : \sigma\,\&\,b$ (and has the same shape).

**Proof** To establish (a), we prove that $C \vdash g_1 \subseteq g_2$ entails $C' \vdash g_1 \subseteq g_2$; this is straightforward by induction. For the claim (b) we proceed by induction on the inference.

For the cases (con), (id) the claim is immediate, and for the cases (abs), (app), (sapp), (let), (rec), (if) it follows directly using the induction hypothesis. For the cases (sub) and (ins) we use (a) together with the induction hypothesis.

**The case (gen).** Then $C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$ because

$$C \cup C_0, A \vdash e : t_0 \,\&\, b$$

$$\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \text{ is well-formed}$$

$$\text{there exists } S \text{ with } \mathit{Dom}(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \text{ such that } C \vdash S\, C_0 \tag{8}$$

$$\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset \tag{9}$$

We now use a small trick: let $R$ be a renaming of the variables of $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C')$ to fresh variables. From $C' \vdash C$ and Lemma 2.18(a) we get $R\,C' \vdash R\,C$ and using (9) we get $R\,C = C$ so $R\,C' \vdash C$. Clearly $R\,C' \cup C_0 \vdash C \cup C_0$ so the induction hypothesis gives $R\,C' \cup C_0, A \vdash e : t_0 \,\&\, b$. Below we verify that

$$\text{there exists } S' \text{ with } \mathit{Dom}(S') \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \text{ such that } R\,C' \vdash S'\, C_0 \tag{10}$$

$$\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(R\,C', A, b) = \emptyset \tag{11}$$

and we then have $R\,C', A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$. Now define the substitution $R'$ such that $\mathit{Dom}(R') = \mathit{Ran}(R)$ and $R'\,\gamma' = \gamma$ if $R\,\gamma = \gamma'$ and $\gamma' \in \mathit{Dom}(R')$. Using Lemma 2.18(b) with the substitution $R'$ we get $C', A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$ as required. Clearly the inference has the same shape.

To prove (10) define $S' = S$. Above we showed that $R\,C' \vdash C$ so using (8) and (a) we get $R\,C' \vdash S'\, C_0$ as required. Finally (11) follows trivially from (9) and $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(R\,C') = \emptyset$. □

# Proof normalisation

**Lemma 2.24** If $A$ is well-formed and solvable from $C$ then an inference tree $C, A \vdash e : \sigma \,\&\, b$ can be transformed into one $C, A \vdash_n e : \sigma \,\&\, b$ that is normalised.

**Proof** We proceed by induction on the inference.

**The case (id). (The case (con) is similar.)** If $A(x)$ is a type then we already have a T-normalised inference. So assume $A(x) = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ and let $R$ be a renaming of $\vec{\alpha}\vec{\beta}\vec{\rho}$ to fresh variables $\vec{\alpha'}\vec{\beta'}\vec{\rho'}$. We can then construct the following TS-normalised inference tree:

$$\dfrac{\dfrac{\dfrac{}{C \cup R\,C_0, A \vdash x : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, \varepsilon}\ (\text{id})}{C \cup R\,C_0, A \vdash x : R\,t_0 \,\&\, \varepsilon}\ (\text{ins})}{C, A \vdash x : \forall(\vec{\alpha'}\vec{\beta'}\vec{\rho'} : R\,C_0).\ R\,t_0 \,\&\, \varepsilon}\ (\text{gen})$$

The rule (ins) is applicable since $Dom(R) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \cup R\,C_0 \vdash R\,C_0$. The rule (gen) is applicable because $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 = \forall(\vec{\alpha'}\vec{\beta'}\vec{\rho'} : R\,C_0).\ R\,t_0$ (up to alpha-renaming) by assumption is well-formed and solvable from $C$, and furthermore $\{\vec{\alpha'}\vec{\beta'}\vec{\rho'}\} \cap FV(C, A, \varepsilon) = \emptyset$ holds by choice of $\vec{\alpha'}\vec{\beta'}\vec{\rho'}$.

**The case (abs).** Then we have $C, A \vdash \texttt{fn}\ x \Rightarrow e : t_1 \to^{\beta} t_2 \,\&\, \varepsilon$ because $C, A[x : t_1] \vdash e : t_2 \,\&\, \beta$. Since $t_1$ is well-formed and solvable from $C$ we can apply the induction hypothesis and get $C, A[x : t_1] \vdash_n e : t_2 \,\&\, \beta$ from which we infer $C, A \vdash_n \texttt{fn}\ x \Rightarrow e : t_1 \to^{\beta} t_2 \,\&\, \varepsilon$.

**The case (app).** Then we have $C, A \vdash e_1\, e_2 : t_1 \,\&\, (b_1; b_2; \beta)$ because $C, A \vdash e_1 : t_2 \to^{\beta} t_1 \,\&\, b_1$ and $C, A \vdash e_2 : t_2 \,\&\, b_2$. Then the induction hypothesis gives $C, A \vdash_n e_1 : t_2 \to^{\beta} t_1 \,\&\, b_1$ and $C, A \vdash_n e_2 : t_2 \,\&\, b_2$. We thus can infer the desired $C, A \vdash_n e_1\, e_2 : t_1 \,\&\, (b_1; b_2; \beta)$.

**The case (let).** Then we have $C, A \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2 \,\&\, (b_1; b_2)$ because $C, A \vdash e_1 : ts_1 \,\&\, b_1$ and $C, A[x : ts_1] \vdash e_2 : t_2 \,\&\, b_2$. Then the induction hypothesis gives $C, A \vdash_n e_1 : ts_1 \,\&\, b_1$. From Fact 2.17 we get that $ts_1$ is well-formed and solvable from $C$, so we can apply the induction hypothesis to get $C, A[x : ts_1] \vdash_n e_2 : t_2 \,\&\, b_2$. This enables us to infer the desired $C, A \vdash_n \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2 \,\&\, (b_1; b_2)$.

**The cases (sapp), (rec), (if), (sub):** Analogous to the above cases.

**The case (ins).** Then $C, A \vdash e : S\,t_0 \,\&\, b$ because with $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ and $C \vdash S\,C_0$ we have $C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$. By applying the induction hypothesis we get

$$C, A \vdash_n e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$$

so by Lemma 2.23 we get $C, A \vdash_n e : S\,t_0 \,\&\, b$ as desired.

**The case (gen).** Then we have $C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$ because $C \cup C_0, A \vdash e : t_0 \,\&\, b$ where $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is well-formed, solvable from $C$ and satisfies $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$. Now $A$ is well-formed and solvable from $C \cup C_0$ (Lemma 2.19) so the induction hypothesis gives $C \cup C_0, A \vdash_n e : t_0 \,\&\, b$. Therefore we have the TS-normalised inference tree $C, A \vdash_n e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b$. □

# Conservative extension

**Theorem 2.25**

Let $e$ be a closed sequential expression $\in$ *Exp*. Let $A$ be defined on sequential constants only and let it behave as in Fig. 2.4; and let $\epsilon(A) = A'$.

- If $A' \vdash_n^{\mathrm{ML}} e : u$ then there exists $\beta$-sequential type $t$ with $\epsilon(t) = u$ such that $C_\beta, A \vdash_n e : t \,\&\, \beta$.

- If $C, A \vdash e : t \,\&\, b$ where $C$ contains no type constraints then there exists an ML type $u$ with $\epsilon(t) = u$ such that $A' \vdash^{\mathrm{ML}} e : u$. □

Before embarking on the proof we need to extend $\epsilon()$ to work on substitutions: from a substitution $S$ we construct an ML substitution $R = \epsilon(S)$ by stipulating $R\,\alpha = \epsilon(S\,\alpha)$.

**Fact A.1** For all substitutions $S$ and types $t$, we have $\epsilon(S\,t) = \epsilon(S)\,\epsilon(t)$.

**Proof** Induction in $t$. If $t = \alpha$, the equation follows from the definition of $\epsilon(S)$. If $t$ is a base type like `int`, the equation is trivial. If $t$ is a composite type like $t_1 \ \rightarrow^{\beta} \ t_2$, the equation reads

$$\epsilon(S\,t_1) \ \rightarrow \ \epsilon(S\,t_2) = \epsilon(S)\,\epsilon(t_1) \ \rightarrow \ \epsilon(S)\,\epsilon(t_2)$$

and follows from the induction hypothesis. If $t$ is a non-sequential type like $t'$ `event` $\beta$, the equation reads $\epsilon(S\,t') = \epsilon(S)\,\epsilon(t')$ which follows from the induction hypothesis. □

## Auxiliary notions I.

Before embarking on the first part of Theorem 2.25 we need to develop some extra machinery; this is due to the fact that the typing of something in *Exp*, such as `tl`$< e >$, may involve the typing of something not in *Exp*, such as `tl`.

**Intermediate expressions.** We say that $e \in EExp$ is an *intermediate expression expecting* $m$ *arguments* if either

- $m = 0$, and $e \in Exp$; or
- $m = 1$, and $e$ is a sequential base function $F_s$; or
- $m > 0$, and $e$ is a constructor $C_s^m$.

Actually we can allow to write $m \geq 0$ in the last clause (cf. Fig. 2.3).

**Non-silent types.** We say that a type is non-silent if it does not contain any subtypes of form $t_1 \ \rightarrow \ t_2$ (but it may contain subtypes of form $t_1 \ \rightarrow^{\beta} \ t_2$).

We say that a type is $m$-order non-silent if it is of the form $t_1 \ \rightarrow \ \cdots t_m \ \rightarrow \ t_0$ with $t_0, t_1, \cdots, t_m$ all non-silent (so to be 0-order non-silent amounts to being non-silent).

We say that a type scheme is ($m$-order) non-silent if its type is.

**Fact A.2** Given ML type $u$, there exists a unique non-silent $\beta$-sequential type $t$ such that $\epsilon(t) = u$.

**Proof** Induction in $u$: if $u = \alpha$ then we can use $t = \alpha$; and there clearly exists no other sequential $t$ with $\epsilon(t) = \alpha$.

Now consider the case where $u$ is a composite type like $u_1 \ \rightarrow \ u_2$. By induction there exists non-silent $\beta$-sequential types $t_1$ and $t_2$ such that $\epsilon(t_1) = u_1$ and $\epsilon(t_2) = u_2$. Let $t = t_1 \ \rightarrow^{\beta} \ t_2$; then $t$ is non-silent and $\beta$-sequential and moreover $\epsilon(t) = u$. Concerning uniqueness, suppose that also $t'$ is non-silent $\beta$-sequential with $\epsilon(t') = u$. From $t'$ being non-silent and sequential we deduce that $t'$ is of form $t'_1 \ \rightarrow^{\beta'} \ t'_2$; as $\epsilon(t'_1) = u_1$ and $\epsilon(t'_2) = u_2$ we from the induction hypothesis deduce that $t'_1 = t_1$ and $t'_2 = t_2$; and from $t'$ being $\beta$-sequential we deduce that $\beta' = \beta$. Hence $t' = t$ as desired. □

## Proof of the first part of Theorem 2.25

The first part of the theorem clearly follows from the following proposition which admits a proof by induction:

**Proposition A.3** Let $e$ be sequential and also an intermediate expression expecting $m$ arguments ($m \geq 0$). Suppose $A' \vdash_n^{\mathrm{ML}} e : u$ with $\epsilon(A) = A'$, where $A(c)$ behaves as in Fig. 2.4 for all sequential constants $c$ and where $A(y)$ is non-silent and $\beta$-sequential for all identifiers $y$ in the domain of $A$.

Then there exists $m$-order non-silent and $\beta$-sequential $t$ with $\epsilon(t) = u$ such that $C_\beta, A \vdash_n e : t \,\&\, \beta$.

Similarly with $us$ and $ts$ instead of $u$ and $t$. □

The proof is by induction on the structure of the normalised proof tree for $A' \vdash_n^{\mathrm{ML}} e : u$ (where the clauses for conditionals and for recursion are omitted, as they present no further complications).

**The case (con):**

Here $u = A'(c)$ and we can use $t = A(c)$; then $\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n c : t \,\&\, \beta$ will follow using (con) and (sub). That $t$ is $m$-order non-silent and $\beta$-sequential follows from an inspection of Fig. 2.4.

**The case (con)(ins):** Here $A' \vdash_n^{\mathrm{ML}} c : R\,u$ holds because $A'(c) = \forall\vec{\alpha}.u$ and $Dom(R) \subseteq \{\vec{\alpha}\}$. Now $A(c)$ takes the form $\forall(\vec{\alpha} : \emptyset).\ t$ with $\epsilon(t) = u$. It is clearly possible (using Fact A.2) to find a substitution $S$ with $Dom(S) \subseteq \{\vec{\alpha}\}$ such that $\epsilon(S) = R$ and such that for all $\alpha \in \{\vec{\alpha}\}$ it holds that $S\,\alpha$ is non-silent and $\beta$-sequential. We can thus use (con), (ins), and (sub) to arrive at the judgement

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n c : S\,t \,\&\, \beta$$

which is as desired since by Fact A.1 we have $\epsilon(S\,t) = R\,u$. Moreover, an inspection of Fig. 2.4 reveals that $t$ is $\beta$-sequential and $m$-order non-silent, from which we deduce that also $S\,t$ is $\beta$-sequential and $m$-order non-silent.

**The case (id):**

Here $u = A'(x)$ and we can use $t = A(x)$; then $\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n x : t \,\&\, \beta$ will follow using (id) and (sub). The assumptions about $A$ tell us that $t$ is non-silent and $\beta$-sequential, and is thus of the desired form since $x$ is an intermediate expression expecting 0 arguments.

**The case (id)(ins):** Here $A' \vdash_n^{\mathrm{ML}} x : R\,u$ holds because $A'(x) = \forall\vec{\alpha}.u$ and $Dom(R) \subseteq \{\vec{\alpha}\}$. Now $A(x)$ takes the form $\forall(\vec{\alpha} : \emptyset).\ t$ with $\epsilon(t) = u$. It is clearly possible (using Fact A.2) to find a substitution $S$ with $Dom(S) \subseteq \{\vec{\alpha}\}$ such that $\epsilon(S) = R$ and such that for all $\alpha \in \{\vec{\alpha}\}$ it holds that $S\,\alpha$ is non-silent and $\beta$-sequential. We can thus use (id), (ins), and (sub) to arrive at the judgement

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n x : S\,t \,\&\, \beta$$

which is as desired since by Fact A.1 we have $\epsilon(S\,t) = R\,u$. The assumptions about $A$ tell us that $t$ is non-silent and $\beta$-sequential, from which we deduce that also $S\,t$ is non-silent and $\beta$-sequential and is thus of the desired form since $x$ is an intermediate expression expecting 0 arguments.

**The case (abs):** As $\texttt{fn}\ x \Rightarrow e \in \mathit{Exp}$ we deduce that also $e \in \mathit{Exp}$. By Fact A.2 there exists non-silent $\beta$-sequential $t_1$ such that $\epsilon(t_1) = u_1$, implying that $\epsilon(A[x : t_1]) = A'[x : u_1]$. We are thus able to apply the induction hypothesis, and we infer that there exists non-silent and $\beta$-sequential $t_2$ with $\epsilon(t_2) = u_2$ such that

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A[x : t_1] \vdash_n e : t_2 \,\&\, \beta.$$

By using (abs) and (sub) we get

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n \texttt{fn}\ x \Rightarrow e : t_1 \ \to^{\beta}\ t_2 \,\&\, \beta$$

which is as desired since $t_1 \ \rightarrow^\beta \ t_2$ is non-silent and $\beta$-sequential and since $\epsilon(t_1 \ \rightarrow^\beta \ t_2) = u_1 \ \rightarrow \ u_2$.

**The case (app):** Clearly $e_1 \ e_2 \in \mathit{Exp}$; and it is easy to see (as $e_1$ is sequential and hence cannot be of the form $F_c$) that it also holds that $e_1, e_2 \in \mathit{Exp}$. We can thus apply the induction hypothesis to find non-silent $\beta$-sequential $t'_1$ and $t'_2$ with $\epsilon(t'_1) = u_2 \ \rightarrow \ u_1$ and $\epsilon(t'_2) = u_2$ such that

$$C_\beta, A \vdash_n \ e_1 \ : \ t'_1 \,\&\, \beta \text{ and } C_\beta, A \vdash_n \ e_2 \ : \ t'_2 \,\&\, \beta.$$

Clearly $t'_1$ takes the form $t_2 \ \rightarrow^\beta \ t_1$, implying $\epsilon(t_1) = u_1$ and $\epsilon(t_2) = u_2$; and as $t_2$ and $t'_2$ are non-silent and $\beta$-sequential we can use Fact A.2 to infer that $t'_2 = t_2$. Hence we can apply (app) to get

$$\{\varepsilon \ \subseteq \ \beta, \ \beta;\beta \ \subseteq \ \beta\}, A \vdash_n \ e_1 \ e_2 \ : \ t_1 \,\&\, (\beta;\beta);\beta$$

so by (sub) we arrive at the desired judgement

$$\{\varepsilon \ \subseteq \ \beta, \ \beta;\beta \ \subseteq \ \beta\}, A \vdash_n \ e_1 \ e_2 \ : \ t_1 \,\&\, \beta.$$

and we have already seen that $t_1$ is of the desired form.

**The case (sapp):** As $e_0 \, @^s_n \, < e_1, \cdots, e_n > \in \mathit{Exp}$ we deduce that $e_0$ is an intermediate expression expecting $n$ arguments and that $e_1, \cdots, e_n \in \mathit{Exp}$. We can thus apply the induction hypothesis to find non-silent and $\beta$-sequential $t_1, t'_1, \cdots, t_n, t'_n, t_0$ such that

$$C_\beta, A \vdash_n \ e_0 \ : \ t'_1 \ \rightarrow \ \cdots t'_n \ \rightarrow \ t_0 \,\&\, \beta \text{ and } \cdots C_\beta, A \vdash_n \ e_i \ : \ t_i \,\&\, \beta \cdots$$

and such that $\epsilon(t_1) = u_1, \cdots, \epsilon(t_n) = u_n$ and such that $\epsilon(t'_1 \ \rightarrow \ \cdots t'_n \ \rightarrow \ t_0)$ $= u_1 \ \rightarrow \ \cdots u_n \ \rightarrow \ u_0$, implying $\epsilon(t'_1) = u_1, \cdots, \epsilon(t'_n) = u_n, \epsilon(t_0) = u_0$. From Fact A.2 we infer that $t'_1 = t_1, \cdots, t'_n = t_n$. Hence we can apply (sapp) to get

$$\{\varepsilon \ \subseteq \ \beta, \ \beta;\beta \ \subseteq \ \beta\}, A \vdash_n \ e_0 \, @^s_n \, < e_1, \cdots, e_n > \ : \ t_0 \,\&\, \beta; \cdots; \beta$$

so by (sub) we arrive at the desired judgement

$$\{\varepsilon \ \subseteq \ \beta, \ \beta;\beta \ \subseteq \ \beta\}, A \vdash_n \ e_0 \, @^s_n \, < e_1, \cdots, e_n > \ : \ t_0 \,\&\, \beta$$

and we have already seen that $t_0$ has the desired properties.

**The case (let):** As $\texttt{let}\ x = e_1\ \texttt{in}\ e_2 \in \mathit{Exp}$ we deduce that also $e_1, e_2 \in \mathit{Exp}$. We can apply the induction hypothesis to find non-silent and $\beta$-sequential $ts_1$ with $\epsilon(ts_1) = us_1$ such that

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n e_1 : ts_1 \,\&\, \beta.$$

Since $\epsilon(A[x : ts_1]) = A'[x : us_1]$ (and $ts_1$ is non-silent and $\beta$-sequential) we can apply the induction hypothesis once more to find non-silent and $\beta$-sequential $t_2$ with $\epsilon(t_2) = u_2$ such that

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A[x : ts_1] \vdash_n e_2 : t_2 \,\&\, \beta.$$

We can now apply (let) and (sub) to get the desired judgement

$$\{\varepsilon \subseteq \beta,\ \beta;\beta \subseteq \beta\}, A \vdash_n \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2 \,\&\, \beta.$$

**The case (gen):** We can apply the induction hypothesis to find $m$-order non-silent and $\beta$-sequential $t$ with $\epsilon(t) = u$ such that

$$C_\beta, A \vdash_n e : t \,\&\, \beta.$$

The conclusion we want to arrive at is

$$C_\beta, A \vdash_n e : \forall(\vec{\alpha} : \emptyset).\, t \,\&\, \beta$$

which follows by using (gen) provided that *(i)* $\forall(\vec{\alpha} : \emptyset).\, t$ is well-formed and solvable from $C_\beta$ and *(ii)* $\{\vec{\alpha}\} \cap FV(C_\beta, A, \beta) = \emptyset$. Here *(i)* is trivial; and *(ii)* follows from $FV(A') \cap \{\vec{\alpha}\} = \emptyset$ since a type variable which is free in $A$ will also be free in $A'$.

## Auxiliary notions II.

Before embarking on the second part of Theorem 2.25 we need to develop some extra machinery.

**ML type equations.** ML type equations are of the form $u_1 = u_2$. With $C_t$ a set of ML type equations and with $R$ an ML substitution, we say that $R$ satisfies (or unifies) $C_t$ iff for all $(u_1 = u_2) \in C_t$ we have $R\,u_1 = R\,u_2$.

The following fact is well-known from unification theory:

**Fact A.4** Let $C_t$ be a set of ML type equations. If there exists an ML substitution which satisfies $C_t$, then $C_t$ has a "most general unifier": that is, an idempotent substitution $R$ which satisfies $C_t$ such that if $R'$ also satisfies $C_t$ then there exists $R''$ such that $R' = R''\,R$.

**Lemma A.5** Suppose $R_0$ with $Dom(R_0) \subseteq G$ satisfies a set of ML type equations $C_t$. Then $C_t$ has a most general unifier $R$ with $Dom(R) \subseteq G$.

**Proof** From Fact A.4 we know that $C_t$ has a most general unifier $R_1$, and hence there exists $R_2$ such that $R_0 = R_2\,R_1$. Let $G_1 = Dom(R_1) \setminus Dom(R_0)$; for $\alpha \in G_1$ we have $R_2\,R_1\,\alpha = R_0\,\alpha = \alpha$ and hence $R_1$ maps the variables in $G_1$ into distinct variables $G_2$ (which by $R_2$ are mapped back again). Since $R_1$ is idempotent we have $G_2 \cap Dom(R_1) = \emptyset$, so $R_0$ equals $R_2$ on $G_2$ showing that $G_2 \subseteq Dom(R_0)$. Moreover, $G_1 \cap G_2 = \emptyset$.

Let $\phi$ map $\alpha \in G_1$ into $R_1\,\alpha$ and map $\alpha \in G_2$ into $R_2\,\alpha$ and behave as the identity otherwise. Then $\phi$ is its own inverse so that $\phi\,\phi = \mathrm{Id}$. Now define $R = \phi\,R_1$; clearly $R$ unifies $C_t$ and if $R'$ also unifies $C_t$ then (since $R_1$ is most general unifier) there exists $R''$ such that $R' = R''\,R_1 = R''\,\phi\,\phi\,R_1 = (R''\,\phi)\,R$.

We are left with showing *(i)* that $R$ is idempotent and *(ii)* that $Dom(R) \subseteq G$. For *(i)*, first observe that $R_1\,\phi$ equals Id except on $Dom(R_1)$. Since $R_1$ is idempotent we have $FV(R_1\,\alpha) \cap Dom(R_1) = \emptyset$ (for all $\alpha$) and hence

$$R\,R = \phi\,R_1\,\phi\,R_1 = \phi\,\mathrm{Id}\,R_1 = R.$$

For *(ii)*, observe that $R$ equals Id on $G_1$ so it will be sufficient to show that $R\,\alpha = \alpha$ if $\alpha \notin (G \cup G_1)$. But then $\alpha \notin Dom(R_0)$ and hence $\alpha \notin G_2$ and $\alpha \notin Dom(R_1)$ so $R\,\alpha = \phi\,\alpha = \alpha$. □

From a constraint set $C$ we construct a set of ML type equations $\epsilon(C)$ as follows:

$$\epsilon(C) = \{(\epsilon(t_1) = \epsilon(t_2)) \mid (t_1 \subseteq t_2) \in C\}.$$

**Fact A.6** Suppose $C \vdash t_1 \subseteq t_2$. If $R$ satisfies $\epsilon(C)$ then $R\,\epsilon(t_1) = R\,\epsilon(t_2)$.

So if $C \vdash C'$ and $R$ satisfies $\epsilon(C)$ then $R$ satisfies $\epsilon(C')$.

**Proof** Induction in the proof tree. If $(t_1 \subseteq t_2) \in C$, the claim follows from the assumptions. The cases for reflexivity and transitivity are straightforward. For the structural rules with the "sequential" type constructors, assume e.g. that $C \vdash t_1 \rightarrow^{\beta} t_2 \subseteq t_1' \rightarrow^{\beta'} t_2'$ because (among other things) $C \vdash t_1' \subseteq t_1$ and $C \vdash t_2 \subseteq t_2'$. By using the induction hypothesis we get the desired equality

$$R\,\epsilon(t_1 \rightarrow^{\beta} t_2) = R\,\epsilon(t_1) \rightarrow R\,\epsilon(t_2) = R\,\epsilon(t_1') \rightarrow R\,\epsilon(t_2') = R\,\epsilon(t_1' \rightarrow^{\beta'} t_2').$$

For the structural rules with the non-sequential type constructors, assume e.g. that $C \vdash t \ \texttt{event}\ \beta \subseteq t' \ \texttt{event}\ \beta'$ because of $C \vdash t \subseteq t'$. Then the desired equality reads $R\,\epsilon(t) = R\,\epsilon(t')$ and follows from the induction hypothesis.

For the backwards rules, assume e.g. that $C \vdash t_1' \subseteq t_1$ holds because of $C \vdash t_1 \rightarrow^{\beta} t_2 \subseteq t_1' \rightarrow^{\beta'} t_2'$. By using the induction hypothesis we have

$$R\,\epsilon(t_1) \rightarrow R\,\epsilon(t_2) = R\,\epsilon(t_1 \rightarrow^{\beta} t_2) = R\,\epsilon(t_1' \rightarrow^{\beta'} t_2') = R\,\epsilon(t_1') \rightarrow R\,\epsilon(t_2')$$

from which the desired relation $R\,\epsilon(t_1) = R\,\epsilon(t_1')$ follows. □

**Relating type schemes.** For a type scheme $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$ we shall not in general (when $C \neq \emptyset$) define any entity $\epsilon(ts)$; this is because one natural attempt, namely $\forall(\vec{\alpha} : \epsilon(C)).\ \epsilon(t)$, is not an ML type scheme and another natural attempt, $\forall\vec{\alpha}.\epsilon(t)$, causes loss of the information in $\epsilon(C)$. Rather we shall define some relations between ML types, types, ML type schemes and type schemes:

**Definition A.7** We write $u \prec_{\epsilon}^{R} ts$, where $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ and where $R$ is an ML substitution, iff there exists $R_0$ which equals $R$ on all variables except $\vec{\alpha}$ such that $R_0$ satisfies $\epsilon(C_0)$ and such that $u = R_0\,\epsilon(t_0)$. □

Notice that instead of demanding $R_0$ to equal $R$ on all variables but $\vec{\alpha}$, it is sufficient to demand that $R_0$ equals $R$ on $FV(ts)$. (We have the expected property that if $u \prec_{\epsilon}^{R} ts$ and $ts$ is alpha-equivalent to $ts'$ then also $u \prec_{\epsilon}^{R} ts'$.)

**Definition A.8** We write $u \prec us$, where $us = \forall\vec{\alpha}.u_0$, iff there exists $R_0$ with $Dom(R_0) \subseteq \vec{\alpha}$ such that $u = R_0\,u_0$.

**Definition A.9** We write $us \cong_{\epsilon}^{R} ts$ to mean that (for all $u$) $u \prec us$ iff $u \prec_{\epsilon}^{R} ts$.

**Fact A.10** Suppose $us = \epsilon(ts)$, where $ts = \forall(\vec{\alpha} : \emptyset).\ t$ is sequential. Then $us \cong^{\mathrm{Id}}_{\epsilon} ts$.

**Proof** We have $us = \forall\vec{\alpha}.\epsilon(t)$, so for any $u$ it holds that $u \prec us \Leftrightarrow \exists\ R$ with $Dom(R) \subseteq \vec{\alpha}$ such that $u = R\,\epsilon(t) \Leftrightarrow u \prec^{\mathrm{Id}}_{\epsilon} ts$. $\Box$

Notice that $\forall().u \cong^{R}_{\epsilon} \forall(() : \emptyset).\ t$ holds iff $u = R\,\epsilon(t)$. We can thus consistently extend $\cong^{R}_{\epsilon}$ to relate not only type schemes but also types:

**Definition A.11** We write $u \cong^{R}_{\epsilon} t$ iff $u = R\,\epsilon(t)$.

**Definition A.12** We write $A' \cong^{R}_{\epsilon} A$ iff $Dom(A') = Dom(A)$ and $A'(x) \cong^{R}_{\epsilon} A(x)$ for all $x \in Dom(A)$.

**Fact A.13** Let $R$ and $S$ be such that $\epsilon(S) = R$. Then the relation $u \prec^{R}_{\epsilon} ts$ holds iff the relation $u \prec^{\mathrm{Id}}_{\epsilon} S\,ts$ holds.

Consequently, $us \cong^{R}_{\epsilon} ts$ holds iff $us \cong^{\mathrm{Id}}_{\epsilon} S\,ts$ holds.

**Proof** Let $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$. Due to the remark after Definition A.7 we can assume that $\vec{\alpha}\vec{\beta}\vec{\rho}$ is disjoint from $Dom(S) \cup Ran(S)$, so $S\,ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : S\,C).\ S\,t$.

First we prove "if". For this suppose that $R'$ equals Id except on $\vec{\alpha}$ and that $R'$ satisfies $\epsilon(S\,C)$ and that $u = R'\,\epsilon(S\,t)$, which by straightforward extensions of Fact A.1 amounts to saying that $R'$ satisfies $R\,\epsilon(C)$ and that $u = R'\,R\,\epsilon(t)$. Since $\{\vec{\alpha}\} \cap Ran(R) = \emptyset$ we conclude that $R'\,R$ equals $R$ except on $\vec{\alpha}$, so we can use $R'\,R$ to show that $u \prec^{R}_{\epsilon} ts$.

Next we prove "only if". For this suppose that $R'$ equals $R$ except on $\vec{\alpha}$ and that $R'$ satisfies $\epsilon(C)$ and that $u = R'\,\epsilon(t)$. Let $R''$ behave as $R'$ on $\vec{\alpha}$ and behave as the identity otherwise. Our task is to show that $R''$ satisfies $\epsilon(S\,C)$ and that $u = R''\,\epsilon(S\,t)$, which as we saw above amounts to showing that $R''$ satisfies $R\,\epsilon(C)$ and that $u = R''\,R\,\epsilon(t)$. This will follow if we can show that $R' = R''\,R$. But if $\alpha \in \vec{\alpha}$ we have $R''\,R\,\alpha = R''\,\alpha = R'\,\alpha$ since $Dom(R) \cap \{\vec{\alpha}\} = \emptyset$, and if $\alpha \notin \vec{\alpha}$ we have $R''\,R\,\alpha = R\,\alpha = R'\,\alpha$ where the first equality sign follows from $Ran(R) \cap \{\vec{\alpha}\} = \emptyset$ and $Dom(R'') \subseteq \vec{\alpha}$. $\Box$

**Fact A.14** If $us \cong^{\mathrm{Id}}_{\epsilon} ts$ then $FV(us) \subseteq FV(ts)$.

**Proof** We assume $us \cong_\epsilon^{\mathrm{Id}} ts$ where $us = \forall\vec{\alpha}\,'.u$ and $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C).\ t$. Let $\alpha_1$ be given such that $\alpha_1 \notin FV(ts)$, our task is to show that $\alpha_1 \notin FV(us)$.

Clearly $u \prec us$ so $u \prec_\epsilon^{\mathrm{Id}} ts$, that is there exists $R$ with $Dom(R) \subseteq \vec{\alpha}$ such that $R$ satisfies $\epsilon(C)$ and such that $u = R\,\epsilon(t)$. Now define a substitution $R_1$ which maps $\alpha_1$ into a fresh variable and is the identity otherwise. Due to our assumption about $\alpha_1$ it is easy to see that $R_1\,R$ equals Id on $FV(ts)$, and as $R_1\,R$ clearly satisfies $\epsilon(C)$ it holds that $R_1\,u = R_1\,R\,\epsilon(t) \prec_\epsilon^{\mathrm{Id}} ts$ and hence also $R_1\,u \prec us$. As $\alpha_1 \notin FV(R_1\,u)$ we can infer the desired $\alpha_1 \notin FV(us)$. □

## Proof of the second part of Theorem 2.25

The second part of the theorem follows (by letting $R = \mathrm{Id}$, employing Fact A.10, and recalling that $A(c)$ is sequential if $c$ is sequential) from the following proposition, which admits a proof by induction.

**Proposition A.15** Let $e \in EExp$ be sequential, suppose $C, A \vdash e : ts \,\&\, b$, suppose $R$ satisfies $\epsilon(C)$, and suppose $A' \cong_\epsilon^R A$; then there exists a $us$ with $us \cong_\epsilon^R ts$ such that $A' \vdash^{\mathrm{ML}} e : us$. Similarly with $t$ and $u$ instead of $ts$ and $us$ (in which case $u = R\,\epsilon(t)$). □

We perform induction in the proof tree for $C, A \vdash e : ts \,\&\, b$, using the terminology from Fig. 2.5 (the clauses for conditionals and for recursion are omitted, as they present no further complications):

**The case (id): (the case (con) is similar)** Suppose $R$ satisfies $\epsilon(C)$, and suppose $A' \cong_\epsilon^R A$. Then $A'(x) \cong_\epsilon^R A(x)$ and $A' \vdash^{\mathrm{ML}} x : A'(x)$, as desired.

**The case (abs):** Suppose $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. Then also $A'[x : R\,\epsilon(t_1)] \cong_\epsilon^R A[x : t_1]$, so the induction hypothesis can be applied to find $u_2$ such that $u_2 = R\,\epsilon(t_2)$ and such that $A'[x : R\,\epsilon(t_1)] \vdash^{\mathrm{ML}} e : u_2$. By using (abs) we get the judgement

$$A' \vdash^{\mathrm{ML}} \texttt{fn}\ x \Rightarrow e : R\,\epsilon(t_1) \rightarrow u_2$$

which is as desired since $R\,\epsilon(t_1) \rightarrow u_2 = R\,\epsilon(t_1 \rightarrow^{\beta} t_2)$.

**The case (app): (the case (sapp) is similar)** Suppose $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. The induction hypothesis can be applied to infer that

$$A' \vdash^{\mathrm{ML}} e_1 : R\,\epsilon(t_2 \to^\beta t_1) \text{ and } A' \vdash^{\mathrm{ML}} e_2 : R\,\epsilon(t_2)$$

and since $R\,\epsilon(t_2 \to^\beta t_1) = R\,\epsilon(t_2) \to R\,\epsilon(t_1)$ we can apply (app) to arrive at the desired judgement $A' \vdash^{\mathrm{ML}} e_1\, e_2 : R\,\epsilon(t_1)$.

**The case (let):** Suppose $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. We can apply the induction hypothesis to find $us_1$ such that $us_1 \cong_\epsilon^R ts_1$ and such that $A' \vdash^{\mathrm{ML}} e_1 : us_1$; and since $A'[x : us_1] \cong_\epsilon^R A[x : ts_1]$ we can apply the induction hypothesis once more to infer that $A'[x : us_1] \vdash^{\mathrm{ML}} e_2 : R\,\epsilon(t_2)$. Now use (let) to arrive at the desired judgement $A' \vdash^{\mathrm{ML}} \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : R\,\epsilon(t_2)$.

**The case (sub):** Suppose $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. By applying the induction hypothesis we infer that $A' \vdash^{\mathrm{ML}} e : R\,\epsilon(t)$ and since by Fact A.6 we have $R\,\epsilon(t) = R\,\epsilon(t')$ this is as desired.

**The case (ins):** Suppose that $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. The induction hypothesis tells us that there exists $us$ with $us \cong_\epsilon^R \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\, t_0$ such that $A' \vdash^{\mathrm{ML}} e : us$.

Since $C \vdash S_0\, C_0$ and $R$ satisfies $\epsilon(C)$, Fact A.6 tells us that $R$ satisfies $\epsilon(S_0\, C_0)$ which by Fact A.1 equals $\epsilon(S_0)\,\epsilon(C_0)$, thus $R\,\epsilon(S_0)$ satisfies $\epsilon(C_0)$. As $R\,\epsilon(S_0)$ equals $R$ except on $\vec{\alpha}$, it holds that $R\,\epsilon(S_0)\,\epsilon(t_0) \prec_\epsilon^R \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\, t_0$ and since $us \cong_\epsilon^R \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\, t_0$ we have $R\,\epsilon(S_0)\,\epsilon(t_0) \prec us$. But this shows that we can use (ins) to arrive at the judgement $A' \vdash^{\mathrm{ML}} e : R\,\epsilon(S_0)\,\epsilon(t_0)$ which is as desired since $\epsilon(S_0)\,\epsilon(t_0) = \epsilon(S_0\, t_0)$ by Fact A.1.

**The case (gen):** Suppose that $R$ satisfies $\epsilon(C)$ and that $A' \cong_\epsilon^R A$. Our task is to find $us$ such that $us \cong_\epsilon^R \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\, t_0$ and such that $A' \vdash^{\mathrm{ML}} e : us$. Below we will argue that we can assume that $\{\vec{\alpha}\} \cap (Dom(R) \cup Ran(R)) = \emptyset$.

> Let $T$ be a renaming substitution mapping $\vec{\alpha}$ into fresh variables $\vec{\alpha}\,'$. By applying Lemma 2.18, by exploiting that $FV(C, A, b) \cap$

$\{\vec{\alpha}\vec{\beta}\vec{\rho}\} = \emptyset$, and by using (gen) we can construct a proof tree whose last nodes are

$$\frac{C \cup T\, C_0, A \vdash e : T\, t_0 \,\&\, b}{C, A \vdash e : \forall(\vec{\alpha}\,'\vec{\beta}\,\vec{\rho} : T\, C_0).\; T\, t_0 \,\&\, b}$$

the conclusion of which is alpha-equivalent to the conclusion of the original proof tree, and the shape of which (by Lemma 2.18) is equal to the shape of the original proof tree.

There exists $S_0$ with $Dom(S_0) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ such that $C \vdash S_0\, C_0$. Fact A.6 then tells us that $R$ satisfies $\epsilon(S_0\, C_0)$ which by Fact A.1 equals $\epsilon(S_0)\, \epsilon(C_0)$.

Now define $R'_0$ to be a substitution with $Dom(R'_0) \subseteq \{\vec{\alpha}\,\}$ which maps $\vec{\alpha}$ into $R\, \epsilon(S_0)\, \vec{\alpha}$. It is easy to see (since $\vec{\alpha}$ is disjoint from $Dom(R) \cup Ran(R)$) that $R'_0\, R = R\, \epsilon(S_0)$, implying that $R'_0$ satisfies $R\, \epsilon(C_0)$.

By Lemma A.5 there exists $R_0$ with $Dom(R_0) \subseteq \{\vec{\alpha}\,\}$ which is a most general unifier of $R\, \epsilon(C_0)$. Hence with $R' = R_0\, R$ it holds not only that $R'$ satisfies $\epsilon(C)$ but also that $R'$ satisfies $\epsilon(C_0)$, so in order to apply the induction hypothesis on $R'$ we just need to show that $A' \cong_{\epsilon}^{R'} A$. This can be done by showing that $R$ equals $R'$ on $FV(A)$, but this follows since our assumptions tell us that $Dom(R_0) \cap FV(R\, A) = \emptyset$.

The induction hypothesis thus tells us that $A' \vdash^{\mathrm{ML}} e : R'\, \epsilon(t_0)$. Let $S$ be such that $\epsilon(S) = R$ and $Dom(S) = Dom(R)$ and $Ran(S) \cap \{\vec{\beta}\,\vec{\rho}\,\} = \emptyset$; since $\{\vec{\alpha}\,\} \cap Ran(R) = \emptyset$ we can also obtain $\{\vec{\alpha}\,\} \cap Ran(S) = \emptyset$. By Fact A.13 and Fact A.14 we infer that $FV(A') \subseteq FV(S\, A)$, so since $\{\vec{\alpha}\,\} \cap FV(A) = \emptyset$ we infer $\{\vec{\alpha}\,\} \cap FV(A') = \emptyset$. We can thus use (gen) to arrive at the judgement $A' \vdash^{\mathrm{ML}} e : \forall \vec{\alpha}\,.R'\, \epsilon(t_0)$.

We are left with showing that $\forall \vec{\alpha}\,.R'\, \epsilon(t_0) \cong_{\epsilon}^{R} \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\; t_0$ but this follows from the following calculation (explained below):

$$\begin{aligned}
& u \prec_{\epsilon}^{R} \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\; t_0 \\
\Leftrightarrow\; & u \prec_{\epsilon}^{\mathrm{Id}} \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : S\, C_0).\; S\, t_0 \\
\Leftrightarrow\; & \exists R_1 \text{ with } Dom(R_1) \subseteq \{\vec{\alpha}\,\} \\
& \qquad \text{such that } R_1 \text{ satisfies } R\, \epsilon(C_0) \text{ and } u = R_1\, R\, \epsilon(t_0) \\
\Leftrightarrow\; & \exists R_1 \text{ with } Dom(R_1) \subseteq \{\vec{\alpha}\,\} \\
& \qquad \text{such that } \exists R_2 : R_1 = R_2\, R_0 \text{ and } u = R_1\, R\, \epsilon(t_0) \\
\Leftrightarrow\; & \exists R_2 \text{ with } Dom(R_2) \subseteq \{\vec{\alpha}\,\} \text{ such that } u = R_2\, R_0\, R\, \epsilon(t_0) \\
\Leftrightarrow\; & u \prec \forall \vec{\alpha}\,.R'\, \epsilon(t_0).
\end{aligned}$$

The first $\Leftrightarrow$ follows from Fact A.13 where we have exploited that $\{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ is disjoint from $Dom(S) \cup Ran(S)$; the second $\Leftrightarrow$ follows from the definition of $\prec_{\epsilon}^{\mathrm{Id}}$ together with Fact A.1; the third $\Leftrightarrow$ is a consequence of $R_0$ being the most general unifier of $R\,\epsilon(C_0)$; and the fourth $\Leftrightarrow$ is a consequence of $Dom(R_0) \subseteq \{\vec{\alpha}\,\}$ since then from $R_1 = R_2\,R_0$ we conclude that if $\alpha' \notin \{\vec{\alpha}\,\}$ then $R_1\,\alpha' = R_2\,\alpha'$ and hence $Dom(R_1) \subseteq \{\vec{\alpha}\,\}$ iff $Dom(R_2) \subseteq \{\vec{\alpha}\,\}$.

# Appendix B

# Proofs of Results Concerning the Semantics

## The sequential semantics

**Fact 3.4** $(E_1[E_2])[e] = E_1[E_2[e]]$.

**Proof** The proof is by induction in $E_1$. If $E_1 = [\,]$ the equation reads $E_2[e] = E_2[e]$, so assume that $E_1$ is a composite context and let us consider the case $E_1 = E\ e_2$ (the other cases are similar). By using the induction hypothesis for $E$ we get the desired equation

$$\begin{aligned} E_1[E_2][e] &= (E\ e_2)[E_2][e] = (E[E_2]\ e_2)[e] = E[E_2][e]\ e_2 \\ &= E[E_2[e]]\ e_2 = E_1[E_2[e]]. \end{aligned}$$

This completes the proof. □

## Reasoning about proof trees

**Fact 3.14** Given $jdg = (C, A \vdash E[e] : \sigma \,\&\, b)$; then there exists (at least one) judgement $jdg'$ of the form $C', A' \vdash e : \sigma' \,\&\, b'$ such that $jdg'$ occurs at $E$ in the inference tree for $jdg$. If $jdg$ is normalised we can assume that $jdg'$

is normalised.

**Proof** The proof is by induction in the inference tree for $jdg$. If $E = [\,]$ we can use $jdg' = jdg$, so assume $E \neq [\,]$. Hence the last rule applied in the inference tree for $jdg$ is none of the following: (con), (id), (abs), or (rec). If (sub), (ins) or (gen) has been applied the induction hypothesis clearly yields the claim; notice that if $jdg$ is normalised then it cannot be the case that (ins) has been applied, as $E[e]$ is neither a constant nor an identifier. So we are left with (app), (sapp), (let) and (if); we only consider (app) as the other cases are similar. Then $E$ takes either the form $E_1\ e_2$ or the form $w_1\ E_2$ or the form $F_c\!<\!E_1\!>$; we consider the former only as the latter are similar.

The situation thus is that $E[e] = E_1[e]\ e_2$ so the left premise of $jdg$ is of the form $C'', A'' \vdash E_1[e] : \sigma'' \,\&\, b''$ (abbreviated $jdg''$). Inductively we can assume that there exists $jdg'$ which occurs at $E_1$ in the inference tree for $jdg''$; but this shows that $jdg'$ occurs at $E$ in the inference tree for $jdg$. $\square$

**Lemma 3.17** Suppose the judgement $jdg' = C', A' \vdash e' : \sigma' \,\&\, b'$ occurs at $E$ with depth $n$ in the inference tree of $jdg = C, A \vdash e : \sigma \,\&\, b$. Then

- $A' = A$;
- if $C$ is well-formed then also $C'$ is well-formed;
- if $C$ is consistent then also $C'$ is consistent.

**Proof** We perform induction in $n$: if $n = 0$ then $C = C'$ and $A = A'$ and the claim is clear.

If $n > 1$ then by Fact 3.15 there exists judgement $jdg'' = C'', A'' \vdash e'' : \sigma'' \,\&\, b''$ and evaluation contexts $E_1$ and $E_2$ such that

> $jdg'$ occurs at $E_1$ with depth $< n$ in the inference tree for $jdg''$; and
> $jdg''$ occurs at $E_2$ with depth $< n$ in the inference tree for $jdg$.

We can thus apply the induction hypothesis twice to infer that $A' = A'' = A$, that if $C$ is well-formed then $C''$ is well-formed and then $C'$ is well-formed; and that if $C$ is consistent then $C''$ is consistent and then $C'$ is consistent.

So we are left with the case $n = 1$, where the inference rule applied is either (app), (sapp), (let) with $jdg'$ as leftmost premise, (if), (sub), (ins) or (gen). In all cases we have $A = A'$; and in all cases but the latter we have $C = C'$. So we only need to consider (gen) where the situation is

$$\frac{jdg' = C \cup C_0, A \vdash e : t_0 \,\&\, b}{jdg = C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}$$

where $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is well-formed (implying that $C_0$ is well-formed) and where $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$ and where there exists $S$ with $Dom(S) \subseteq \{\vec{\alpha}\vec{\beta}\vec{\rho}\}$ such that $C \vdash S\,C_0$, implying that $C \vdash S\,(C \cup C_0)$. We need to show that if $C$ is consistent then $C \cup C_0$ is consistent: but if $C \cup C_0 \vdash t_1 \subseteq t_2$ where there is a mismatch between $t_1$ and $t_2$ then (by Lemma 2.18 and 2.19) $C \vdash S\,t_1 \subseteq S\,t_2$ and as there clearly is a mismatch between $S\,t_1$ and $S\,t_2$ this conflicts with our assumption about $C$ being consistent. □

**Lemma 3.20** Suppose that $C, A[x : \sigma'] \vdash_n e : \sigma \,\&\, b$ and that $C, A \vdash_n e' : \sigma' \,\&\, \varepsilon$; then $C, A \vdash_n e[e'/x] : \sigma \,\&\, b$.

**Proof** Induction in the shape of the proof tree for $C, A[x : \sigma'] \vdash_n e : \sigma \,\&\, b$; we perform case analysis on the way it is constructed (cf. Definition 2.22).

**(con) or (con)(ins) has been applied:** Then $e$ is a constant, and $e[e'/x] = e$ so the claim is clear.

**(id) or (id)(ins) has been applied:** Then $e$ is an identifier $y$. If $y \neq x$ then $e[e'/x] = e$ and the claim is clear since $A[x : \sigma'](y) = A(y)$.

If $y = x$ then the inference takes the form

$$\frac{C, A[x : \sigma'] \vdash x : \sigma' \,\&\, \varepsilon}{C, A[x : \sigma'] \vdash x : t \,\&\, \varepsilon}$$

where the last rule follows by zero or one application of (ins). We must show

$$C, A \vdash_n e' : t \,\&\, \varepsilon$$

but this follows from the second part of the assumption, using Lemma 2.23.

**(abs) has been applied:** Here the inference takes the form

$$\frac{C, A[x : \sigma'][y : t_1] \vdash_n e : t_2 \,\&\, \beta}{C, A[x : \sigma'] \vdash_n \texttt{fn}\ y{\Rightarrow}e : t_1 \ \to^{\beta}\ t_2 \,\&\, \varepsilon}$$

where we can assume (by suitable alpha-renaming) that $y \neq x$ and that $y$ does not occur in $e'$. Hence we can apply Fact 2.20 and Fact 2.21 to get

> $C, A[y : t_1][x : \sigma'] \vdash_n e : t_2 \,\&\, \beta$ with the same shape as the premise
> $C, A[y : t_1] \vdash_n e' : \sigma' \,\&\, \varepsilon$.

We can thus apply the induction hypothesis and subsequently use (abs) to construct an inference tree whose last inference is

$$\frac{C, A[y : t_1] \vdash_n e[e'/x] : t_2 \,\&\, \beta}{C, A \vdash_n \texttt{fn}\ y{\Rightarrow}e[e'/x] : t_1 \ \to^{\beta}\ t_2 \,\&\, \varepsilon}$$

which is as desired since $(\texttt{fn}\ y{\Rightarrow}e)[e'/x] = (\texttt{fn}\ y{\Rightarrow}e[e'/x])$.

**(app) has been applied:** Here the inference takes the form

$$\frac{C, A[x : \sigma'] \vdash_n e_1 : t_2 \ \to^{\beta}\ t_1 \,\&\, b_1 \qquad C, A[x : \sigma'] \vdash_n e_2 : t_2 \,\&\, b_2}{C, A[x : \sigma'] \vdash_n e_1\ e_2 : t_1 \,\&\, (b_1; b_2; \beta)}$$

where we can apply the induction hypothesis twice and subsequently use (app) to construct an inference tree whose last inference is

$$\frac{C, A \vdash_n e_1[e'/x] : t_2 \ \to^{\beta}\ t_1 \,\&\, b_1 \qquad C, A \vdash_n e_2[e'/x] : t_2 \,\&\, b_2}{C, A \vdash_n e_1[e'/x]\ e_2[e'/x] : t_1 \,\&\, (b_1; b_2; \beta)}$$

which is as desired since $(e_1\ e_2)[e'/x] = e_1[e'/x]\ e_2[e'/x]$.

**(sapp), (let), (rec) or (if) has been applied:** Similar to the above two cases, exploiting Fact 2.20 and Fact 2.21 and we only spell the case (rec) out in detail. Here the inference takes the form

$$\frac{C, A[x : \sigma'][f : t] \vdash_n \texttt{fn}\ y{\Rightarrow}e : t \,\&\, b}{C, A[x : \sigma'] \vdash_n \texttt{rec}\ f\ y{\Rightarrow}e : t \,\&\, b}$$

where we can assume that $y \neq x$, $f \neq x$ and that neither $y$ nor $f$ occurs in $e'$. Hence we can apply Fact 2.20 and Fact 2.21 to get

$$C, A[f : t][x : \sigma'] \vdash_n \texttt{fn}\ y{\Rightarrow}e \ : \ t \,\&\, b$$
$$C, A[f : t] \vdash_n e' \ : \ \sigma' \,\&\, \varepsilon.$$

and as the former inference has the same shape as the premise we can apply the induction hypothesis to infer

$$C, A[f : t] \vdash_n (\texttt{fn}\ y{\Rightarrow}e)[e'/x] \ : \ t \,\&\, b$$

which since $y \neq x$ and $y$ does not occur in $e'$ amounts to

$$C, A[f : t] \vdash_n \texttt{fn}\ y{\Rightarrow}e[e'/x] \ : \ t \,\&\, b.$$

By applying (rec) we get

$$C, A \vdash_n \texttt{rec}\ f\ y{\Rightarrow}e[e'/x] \ : \ t \,\&\, b$$

which is as desired since $(\texttt{rec}\ f\ y{\Rightarrow}e)[e'/x] = (\texttt{rec}\ f\ y{\Rightarrow}e[e'/x])$.

**(sub) has been applied:** Here the inference takes the form

$$\frac{C, A[x : \sigma'] \vdash_n e \ : \ t \,\&\, b}{C, A[x : \sigma'] \vdash_n e \ : \ t' \,\&\, b'}$$

with $C \vdash t \subseteq t'$ and $C \vdash b \subseteq b'$ so we can apply the induction hypothesis and subsequently use (sub) to construct an inference tree whose last inference is

$$\frac{C, A \vdash_n e[e'/x] \ : \ t \,\&\, b}{C, A \vdash_n e[e'/x] \ : \ t' \,\&\, b'}$$

**(gen) has been applied:** Here the inference takes the form

$$\frac{C \cup C_0, A[x : \sigma'] \vdash_n e \ : \ t_0 \,\&\, b}{C, A[x : \sigma'] \vdash_n e \ : \ ts \,\&\, b}$$

where $ts = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is well-formed, solvable from $C$, and satisfies $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A[x : \sigma'], b) = \emptyset$. By Lemma 2.19 we have

$$C \cup C_0, A \vdash_n e' \ : \ \sigma' \,\&\, \varepsilon$$

so we can apply the induction hypothesis to get

$$C \cup C_0, A \vdash_n e[e'/x] : t_0 \,\&\, b.$$

We can then apply (gen) (since $\{\vec{\alpha}\vec{\beta}\vec{\rho}\} \cap FV(C, A, b) = \emptyset$) to arrive at the desired judgement $C, A \vdash_n e[e'/x] : ts \,\&\, b$. □

**Lemma 3.21** Suppose that $C, A \vdash_n w : \sigma \,\&\, b$; then

- $C \vdash \varepsilon \subseteq b$ and
- $C, A \vdash_n w : \sigma \,\&\, \varepsilon$.

**Proof** It is enough to consider the case where $\sigma$ is a type $t$, for if the inference

$$\frac{C \cup C_0, A \vdash w : t_0 \,\&\, b}{C, A \vdash w : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}\ \text{(gen)}$$

is valid it remains valid when $b$ is replaced by $\varepsilon$. We now prove the claim by induction in the size of $w$, and the only interesting case is where $w = C^n < w_1, \cdots, w_n >$ with $n \geq 1$.

The normalised inference takes the form

$$\frac{C, A \vdash_n C^n : t_1 \rightarrow \cdots t_n \rightarrow t_0 \,\&\, b_0 \cdots C, A \vdash_n w_i : t_i \,\&\, b_i \cdots}{C, A \vdash_n C^n < w_1, \cdots, w_n > : t \,\&\, b}\text{(sapp),(sub)}^*$$

where $C \vdash t_0 \subseteq t$ and $C \vdash b_0; b_1; \cdots; b_n \subseteq b$. We now infer, making use of the induction hypothesis for $w_1 \cdots w_n$, *(i)* that $C \vdash \varepsilon \subseteq b_i$ for $i \in \{0 \cdots n\}$, implying $C \vdash \varepsilon \subseteq \varepsilon; \varepsilon; \cdots; \varepsilon \subseteq b_0; b_1; \cdots; b_n \subseteq b$; *(ii)* that we can construct the inference tree

$$\frac{C, A \vdash_n C^n : t_1 \rightarrow \cdots t_n \rightarrow t_0 \,\&\, \varepsilon \cdots C, A \vdash_n w_i : t_i \,\&\, \varepsilon \cdots}{C, A \vdash_n C^n < w_1, \cdots, w_n > : t \,\&\, \varepsilon}\text{(sapp),(sub)}$$

□

**Lemma 3.24** Suppose the judgement $jdg' = (C', A \vdash e : \sigma' \,\&\, b')$ occurs at $E$ with depth $n'$ in the normalised inference $jdg = (C, A \vdash_n E[e] : \sigma \,\&\, b)$ where $C$ (and by Lemma 3.17 then also $C'$) is well-formed and consistent.

Let $b_n$ be a behaviour and let $A_n$ be of the form $A[x_1 : \sigma_1][\cdots : \cdots][x_m : \sigma_m]$ with $m \geq 0$, such that $x_1 \cdots x_m$ do not occur in $E[e]$ and such that $FV(\sigma_1) \cup \cdots \cup FV(\sigma_m) \subseteq FV(b_n)$.

Let $e_n$ be an expression and $b'_r$ a behaviour such that

$$C', A_n \vdash_n e_n : \sigma' \,\&\, b'_r \text{ and}$$
$$C' \vdash b_n; b'_r \subseteq b'.$$

Then there exists $b_r$ such that

$$C, A_n \vdash_n E[e_n] : \sigma \,\&\, b_r \text{ and}$$
$$C \vdash b_n; b_r \subseteq b.$$

Moreover, there exists $S$ with $Dom(S) \cap FV(A, b_n) = \emptyset$ such that $C \vdash S\, C'$.

**Proof** We perform induction in $n'$: if $n' = 0$ then $E = [\,]$, $C = C'$, $\sigma = \sigma'$, $b = b'$ and the claim is trivial as we can choose $b_r = b'_r$ and $S = \text{Id}$.

If $n' > 1$ then by Fact 3.15 there exists evaluation contexts $E_1$ and $E_2$ with $E = E_2[E_1]$ and judgement $jdg'' = C'', A \vdash_n e'' : \sigma'' \,\&\, b''$ such that

> $jdg'$ occurs at $E_1$ with depth $< n'$ in the inference tree for $jdg''$; and
> $jdg''$ occurs at $E_2$ with depth $< n'$ in the inference tree for $jdg$.

By Lemma 3.17 $C''$ is well-formed and consistent, so if $C', A_n \vdash_n e_n : \sigma' \,\&\, b'_r$ and $C' \vdash b_n; b'_r \subseteq b'$ we can apply the induction hypothesis (with $jdg'$ and $jdg''$) to infer that there exists $b''_r$ and $S_1$ such that $C'', A_n \vdash_n E_1[e_n] : \sigma'' \,\&\, b''_r$ and $C'' \vdash b_n; b''_r \subseteq b''$ and $Dom(S_1) \cap FV(A, b_n) = \emptyset$ and $C'' \vdash S_1\, C'$. We can then apply the induction hypothesis once more (with $jdg''$ and $jdg$) to infer that there exists $b_r$ and $S_2$ such that $C, A_n \vdash_n E_2[E_1[e_n]] : \sigma \,\&\, b_r$ and $C \vdash b_n; b_r \subseteq b$ and $Dom(S_2) \cap FV(A, b_n) = \emptyset$ and $C \vdash S_2\, C''$. This is as desired, since with $S = S_2\, S_1$ we have $Dom(S) \cap FV(A, b_n) = \emptyset$ and (by Lemma 2.18 and 2.19) $C \vdash S\, C'$.

So we are left with the case $n' = 1$. We perform case analysis on $E$:

$\underline{E = E_1\ e_2}$: Here $E_1 = [\,]$ and the situation is:

$$\frac{jdg' = C, A \vdash_n e_1 : (t_2 \rightarrow^{\beta} t_1) \,\&\, b_1 \qquad C, A \vdash_n e_2 : t_2 \,\&\, b_2}{jdg = C, A \vdash e_1\ e_2 : t_1 \,\&\, (b_1; b_2; \beta)}$$

and our assumptions are

$$C, A_n \vdash_n e_n : t_2 \rightarrow^{\beta} t_1 \,\&\, b_r' \text{ and}$$
$$C \vdash b_n; b_r' \subseteq b_1$$

and we must show that there exists $b_r$ and $S$ such that

$$C, A_n \vdash_n e_n\, e_2 : t_1 \,\&\, b_r \qquad (1)$$
$$C \vdash b_n; b_r \subseteq b_1; b_2; \beta \qquad (2)$$
$$Dom(S) \cap FV(A, b_n) = \emptyset \text{ and } C \vdash S\,C.$$

We can choose $b_r = b_r'; b_2; \beta$ and $S = \text{Id}$: then (2) is a trivial consequence of the assumptions and of $\subseteq$ being a congruence; and (1) will follow provided we can show that

$$C, A_n \vdash_n e_2 : t_2 \,\&\, b_2$$

but this follows from Fact 2.21.

$\underline{E = w\ E_2}$: Here $E_2 = [\,]$ and the situation is:

$$\frac{C, A \vdash_n w : (t_2 \rightarrow^{\beta} t_1) \,\&\, b_1 \qquad jdg' = C, A \vdash_n e_2 : t_2 \,\&\, b_2}{jdg = C, A \vdash w\ e_2 : t_1 \,\&\, (b_1; b_2; \beta)}$$

and our assumptions are

$$C, A_n \vdash_n e_n : t_2 \,\&\, b_r' \text{ and}$$
$$C \vdash b_n; b_r' \subseteq b_2$$

and we must show that there exists $b_r$ and $S$ such that

$$C, A_n \vdash_n w\ e_n : t_1 \,\&\, b_r \text{ and}$$
$$C \vdash b_n; b_r \subseteq b_1; b_2; \beta \text{ and}$$
$$Dom(S) \cap FV(A, b_n) = \emptyset \text{ and } C \vdash S\,C.$$

By Lemma 3.21 and Fact 2.21 we infer that

$$C \vdash \varepsilon \subseteq b_1 \text{ and } C, A_n \vdash_n w : (t_2 \rightarrow^\beta t_1) \,\&\, \varepsilon$$

which shows than we can use $b_r = b'_r; \beta$ and trivially $S = \mathrm{Id}$.

<u>$E = \texttt{let } x = E_1 \texttt{ in } e_2$:</u> Here $E_1 = [\,]$ and the situation is:

$$\frac{jdg' = C, A \vdash_n e_1 : ts_1 \,\&\, b_1 \qquad C, A[x : ts_1] \vdash_n e_2 : t_2 \,\&\, b_2}{jdg = C, A \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : t_2 \,\&\, (b_1; b_2)}$$

and our assumptions are

$$C, A_n \vdash_n e_n : ts_1 \,\&\, b'_r \text{ and}$$
$$C \vdash b_n; b'_r \subseteq b_1$$

and we must show that there exists $b_r$ and $S$ such that

$$C, A_n \vdash_n \texttt{let } x = e_n \texttt{ in } e_2 : t_2 \,\&\, b_r \tag{3}$$
$$C \vdash b_n; b_r \subseteq b_1; b_2 \tag{4}$$
$$\mathit{Dom}(S) \cap \mathit{FV}(A, b_n) = \emptyset \text{ and } C \vdash S\,C.$$

We can choose $b_r = b'_r; b_2$ and $S = \mathrm{Id}$: then (4) is a trivial consequence of the assumptions and of $\subseteq$ being a congruence; and (3) will follow provided we can show that

$$C, A_n[x : ts_1] \vdash_n e_2 : t_2 \,\&\, b_2.$$

But this follows from Fact 2.21 and Fact 2.20 since all of $x_1 \cdots x_m$ are $\neq x$ and do not occur in $e_2$.

<u>$E = \texttt{if } E_0 \texttt{ then } e_1 \texttt{ else } e_2$:</u> Here $E_0 = [\,]$ and the situation is:

$$\frac{jdg' = C, A \vdash_n e_0 : \texttt{bool} \,\&\, b_0 \qquad C, A \vdash_n e_1 : t \,\&\, b_1 \qquad C, A \vdash_n e_2 : t \,\&\, b_2}{jdg = C, A \vdash \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 : t \,\&\, b_0; (b_1 + b_2)}$$

and our assumptions are

$C, A_n \vdash_n e_n : \texttt{bool} \,\&\, b'_r$ and
$C \vdash b_n; b'_r \subseteq b_0$

and we must show that there exists $b_r$ and $S$ such that

$C, A_n \vdash_n \texttt{if}\ e_n\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t \,\&\, b_r$ and
$C \vdash b_n; b_r \subseteq b_0; (b_1 + b_2)$ and
$Dom(S) \cap FV(A, b_n) = \emptyset$ and $C \vdash S\, C$.

We can choose $b_r = b'_r; (b_1 + b_2)$ and $S = \mathrm{Id}$; then the claims will follow since by Fact 2.21 we have

$C, A_n \vdash_n e_1 : t \,\&\, b_1$ and $C, A_n \vdash_n e_2 : t \,\&\, b_2$.

<u>$E = F_s\!<\!E_1\!>$:</u> Here $E_1 = [\,]$ and the situation is:

$$\frac{C, A \vdash_n F_s : (t_1 \rightarrow t_0) \,\&\, b_0 \qquad jdg' = C, A \vdash_n e_1 : t_1 \,\&\, b_1}{jdg = C, A \vdash F_s\!<\!e_1\!> : t_0 \,\&\, (b_0; b_1)}$$

and our assumptions are

$C, A_n \vdash_n e_n : t_1 \,\&\, b'_r$ and
$C \vdash b_n; b'_r \subseteq b_1$

and we must show that there exists $b_r$ and $S$ such that

$C, A_n \vdash_n F_s\!<\!e_n\!> : t_0 \,\&\, b_r$ and
$C \vdash b_n; b_r \subseteq b_0; b_1$ and
$Dom(S) \cap FV(A, b_n) = \emptyset$ and $C \vdash S\, C$.

We clearly have

$C \vdash \varepsilon \subseteq b_0$ and $C, A_n \vdash_n F_s : (t_1 \rightarrow t_0) \,\&\, \varepsilon$

which shows than we can use $b_r = b'_r$ and trivially $S = \mathrm{Id}$.

<u>$E = F_c\!<\!E_1\!>$:</u> This case is much similar to the case $E = w\ E_2$.

$\underline{E = C^p\!<\cdots, w_{i-1}, E_i, e_{i+1}, \cdots\!>}$**:** Here $E_i = [\,]$ and the situation is that

$$C, A \vdash_n \ C^p\!<\!w_1, \cdots, w_{i-1}, e_i, e_{i+1}, \cdots, e_p\!> \ : \ t_0 \,\&\, b_0; b_1; \cdots; b_p$$

because

$$\begin{aligned}
&C, A \vdash_n \ C^p \ : \ (t_1 \ \rightarrow \ \cdots t_p \ \rightarrow \ t_0) \,\&\, b_0 \text{ and}\\
&C, A \vdash_n \ w_j \ : \ t_j \,\&\, b_j \text{ for all } j \in \{1 \cdots i \Leftrightarrow 1\} \text{ and}\\
&C, A \vdash_n \ e_j \ : \ t_j \,\&\, b_j \text{ for all } j \in \{i \cdots p\}.
\end{aligned}$$

Our assumptions are

$$\begin{aligned}
&C, A_n \vdash_n \ e_n \ : \ t_i \,\&\, b'_r \text{ and}\\
&C \vdash \ b_n; b'_r \ \subseteq \ b_i
\end{aligned}$$

and we must show that there exists $b_r$ and $S$ such that

$$\begin{aligned}
&C, A_n \vdash_n \ C^p\!<\!w_1, \cdots, w_{i-1}, e_n, e_{i+1}, \cdots, e_p\!> \ : \ t_0 \,\&\, b_r \text{ and}\\
&C \vdash \ b_n; b_r \ \subseteq \ b_0; b_1; \cdots; b_p \text{ and}\\
&\mathit{Dom}(S) \cap FV(A, b_n) = \emptyset \text{ and } C \vdash \ S\,C.
\end{aligned}$$

We infer (making use of Lemma 3.21) that

$$C \vdash \ \varepsilon \ \subseteq \ b_j \text{ for all } j \in \{0 \cdots i \Leftrightarrow 1\}$$

and using Fact 2.21 and Lemma 3.21 we infer that

$$\begin{aligned}
&C, A_n \vdash_n \ C^p \ : \ (t_1 \ \rightarrow \ \cdots t_p \ \rightarrow \ t_0) \,\&\, \varepsilon \text{ and}\\
&C, A_n \vdash_n \ w_j \ : \ t_j \,\&\, \varepsilon \text{ for all } j \in \{1 \cdots i \Leftrightarrow 1\} \text{ and}\\
&C, A_n \vdash_n \ e_j \ : \ t_j \,\&\, b_j \text{ for all } j \in \{i+1 \cdots p\}
\end{aligned}$$

which shows that we can use $b_r = b'_r; b_{i+1}; \cdots; b_p$ and trivially $S = \mathrm{Id}$.

$\underline{E = [\,]}$**:** In this case $jdg$ follows from $jdg'$ by one application of either (sub), (ins) or (gen).

**(sub) has been applied:** the situation is

$$\frac{jdg' = C, A \vdash_n e : t \,\&\, b}{jdg = C, A \vdash e : t' \,\&\, b'}$$

where $C \vdash t \subseteq t'$ and $C \vdash b \subseteq b'$. Our assumptions are

$C, A_n \vdash_n e_n : t \,\&\, b'_r$ and
$C \vdash b_n; b'_r \subseteq b$

and we must show that there exists $b_r$ and $S$ such that

$C, A_n \vdash_n e_n : t' \,\&\, b_r$ and
$C \vdash b_n; b_r \subseteq b'$ and
$Dom(S) \cap FV(A, b_n) = \emptyset$ and $C \vdash S\,C$.

But we can clearly choose $b_r = b'_r$ and $S = \mathrm{Id}$.

**(ins) has been applied:** the situation is

$$\frac{jdg' = C, A \vdash e : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b}{jdg = C, A \vdash e : S_0\, t_0 \,\&\, b}$$

where $\forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$ is solvable from $C$ by $S_0$ (and where the premise is constructed by (con) or (id)). Our assumptions are

$C, A_n \vdash_n e_n : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, b'_r$ and
$C \vdash b_n; b'_r \subseteq b$

and we must show that there exists $b_r$ and $S$ such that

$C, A_n \vdash_n e_n : S_0\, t_0 \,\&\, b_r$ and
$C \vdash b_n; b_r \subseteq b$ and
$Dom(S) \cap FV(A, b_n) = \emptyset$ and $C \vdash S\,C$.

But we can clearly choose $b_r = b'_r$ and $S = \mathrm{Id}$, using Lemma 2.23.

**(gen) has been applied:** this case has been covered in the main text. □

# Appendix C

# Proofs of Results Concerning the Algorithm

## Algorithm $\mathcal{R}$

**Lemma 4.26** Suppose $A \vdash (C', t', b') \Leftrightarrow\!\!\!\rightarrow (C'', t'', b'')$ and let $\gamma_1, \gamma_2 \in FV(C'')$. Then $(\gamma_1 \Leftarrow^* \gamma_2) \in C'$ holds iff $(\gamma_1 \Leftarrow^* \gamma_2) \in C''$ holds.

**Proof** (We use the terminology from the relevant clauses in Figure 4.5, which does not conflict with the one used in the formulation of the lemma.) For (redund) this is a straightforward consequence of the assumptions. For (cycle), (shrink) and (boost) the "only if"-part follows from Fact 4.14: if $(\gamma_1 \Leftarrow^* \gamma_2) \in C'$ then $(S\,\gamma_1 \Leftarrow^* S\,\gamma_2) \in S\,C'$ and as $\gamma_1, \gamma_2 \notin \mathit{Dom}(S)$ this amounts to $(\gamma_1 \Leftarrow^* \gamma_2) \in S\,C'$ which is clearly equivalent to $(\gamma_1 \Leftarrow^* \gamma_2) \in C''$.

We are left with proving the "if"-part for (cycle), (shrink) and (boost); to do so it suffices to show that

$$(\gamma_1' \subseteq \gamma_2') \in C'' \text{ implies } (\gamma_1' \Leftarrow^* \gamma_2') \in C'.$$

As $C'' = S\,C$ we can assume that there exists $(\gamma_1 \subseteq \gamma_2) \in C$ such that $\gamma_1' = S\,\gamma_1$ and $\gamma_2' = S\,\gamma_2$; then (since $C \subseteq C'$) our task can be accomplished by showing that

$$(S\,\gamma_1 \Leftarrow^* \gamma_1) \in C' \text{ and } (\gamma_2 \Leftarrow^* S\,\gamma_2) \in C'.$$

This is trivial except if $\gamma_1 = \gamma$ or $\gamma_2 = \gamma$. The former is impossible in the case (boost) (as $LHS(C)$ is anti-monotonic in $\gamma$) and otherwise the claim follows from the assumptions; the latter is impossible in the case (shrink) (as $\gamma \notin RHS(C)$) and otherwise the claim follows from the assumptions. □

**Lemma C.1** Let $S = [\gamma \mapsto \gamma']$.

1. If $\gamma_1 \in M(g)$ then $\gamma_1 \in M(S\,g)$, provided that $\gamma \in M(g)$ or $\gamma_1 \neq \gamma'$.

2. If $\gamma_1 \in A(g)$ then $\gamma_1 \in A(S\,g)$, provided that $\gamma \in A(g)$ or $\gamma_1 \neq \gamma'$.

**Proof** Induction in $g$. First consider the case where $g$ is a variable: then also $S\,g$ is a variable so (1) follows vacuously; for (2) we must show that if $\gamma_1 \neq g$ then $\gamma_1 \neq S\,g$, but this follows from the side condition which reads $\gamma \neq g$ or $\gamma_1 \neq \gamma'$.

Next consider the case where $g$ is a function type $t_1 \rightarrow^{\beta} t_2$. Let $\gamma_1 \in M(g)$ (the case $\gamma_1 \in A(g)$ is rather similar) and let $\gamma \in M(g)$ or $\gamma_1 \neq \gamma'$, then $\gamma_1 \in A(t_1) \cap M(t_2)$ and we also have $\gamma \in A(t_1) \cap M(t_2)$ or $\gamma_1 \neq \gamma'$. Thus we can apply the induction hypothesis to infer (by 2) that $\gamma_1 \in A(S\,t_1)$ and to infer (by 1) that $\gamma_1 \in M(S\,t_2)$; hence $\gamma_1 \in M(S\,t_1 \rightarrow^{S\,\beta} S\,t_2) = M(S\,g)$ as desired.

Next consider the case where $g$ is a behaviour $\rho\,!\,t$. Let $\gamma_1 \in M(g)$ (the case $\gamma_1 \in A(g)$ is similar), then $\gamma_1 \notin FV(g)$. If $\gamma \in M(g)$ then $\gamma \notin FV(g)$ so $S\,g = g$ and the claim is trivial; if $\gamma_1 \neq \gamma'$ then $\gamma_1 \notin FV(S\,g)$ so $\gamma_1 \in M(S\,g)$.

The other cases are similar. □

**Proposition 4.28** Suppose that

$$A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C_1, t_1, b_1) \text{ and}$$
$$A \vdash (C, t, b) \Leftrightarrow\!\!\!\rightarrow (C_2, t_2, b_2)$$

where $C$ is *acyclic* as well as atomic. Then there exists $(C_1', t_1', b_1')$ and $(C_2', t_2', b_2')$, which are equal up to renaming, such that

$$A \vdash (C_1, t_1, b_1) \Leftrightarrow\!\!\!\rightarrow^{\leq 1} (C_1', t_1', b_1') \text{ and}$$
$$A \vdash (C_2, t_2, b_2) \Leftrightarrow\!\!\!\rightarrow^{\leq 1} (C_2', t_2', b_2').$$

**Proof** As (cycle) is not applicable, each of the two rewriting steps in the assumption can be of three kinds yielding six different combinations:

**(redund) and (redund)** eliminating $(\gamma_1' \subseteq \gamma_1)$ and $(\gamma_2' \subseteq \gamma_2)$ where we can assume that either $\gamma_1' \neq \gamma_2'$ or $\gamma_1 \neq \gamma_2$ as otherwise the claim is trivial. The situation thus is

$$A \vdash (C \dot\cup \{\gamma_1' \subseteq \gamma_1\} \dot\cup \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\rightarrow (C \dot\cup \{\gamma_2' \subseteq \gamma_2\}, t, b)$$
$$A \vdash (C \dot\cup \{\gamma_1' \subseteq \gamma_1\} \dot\cup \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\rightarrow (C \dot\cup \{\gamma_1' \subseteq \gamma_1\}, t, b)$$

where

$$(\gamma_1' \Leftarrow^* \gamma_1) \in C \dot\cup \{\gamma_2' \subseteq \gamma_2\} \text{ and} \tag{1}$$
$$(\gamma_2' \Leftarrow^* \gamma_2) \in C \dot\cup \{\gamma_1' \subseteq \gamma_1\}. \tag{2}$$

It will suffice to show that

$$\text{either } (\gamma_1' \Leftarrow^* \gamma_1) \in C \text{ or } (\gamma_2' \Leftarrow^* \gamma_2) \in C \tag{3}$$

for if e.g. $(\gamma_1' \Leftarrow^* \gamma_1) \in C$ holds then by (2) also $(\gamma_2' \Leftarrow^* \gamma_2) \in C$ holds and we can apply (redund) twice to complete the diamond.

For the sake of arriving at a contradiction we now assume that (3) does not hold. Using (1) and (2) we see that the situation is that

$$(\gamma_1' \Leftarrow^* \gamma_2') \in C \text{ and } (\gamma_2 \Leftarrow^* \gamma_1) \in C \text{ and}$$
$$(\gamma_2' \Leftarrow^* \gamma_1') \in C \text{ and } (\gamma_1 \Leftarrow^* \gamma_2) \in C$$

and this conflicts with the assumption about the graph being cycle-free.

**(redund) and (shrink)** eliminating $(\gamma_1' \subseteq \gamma_1)$ and shrinking $\gamma_2$ into $\gamma_2'$ (with $\gamma_2' \neq \gamma_2$). First notice that it cannot be the case that $(\gamma_1' \subseteq \gamma_1) = (\gamma_2' \subseteq \gamma_2)$, for then we would have $(\gamma_1' \Leftarrow^* \gamma_1) \in C$ as well as $\gamma_2 \notin \mathit{RHS}(C)$ (with $C$ the remaining constraints). The situation thus is

$$
\begin{aligned}
A \vdash\ & (C \,\dot\cup\, \{\gamma_1' \subseteq \gamma_1\} \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\!\rightarrow \\
& (C \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \\
A \vdash\ & (C \,\dot\cup\, \{\gamma_1' \subseteq \gamma_1\} \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\!\rightarrow \\
& (S\,C \cup \{S\,\gamma_1' \subseteq S\,\gamma_1\}, S\,t, S\,b)
\end{aligned}
$$

where $S = [\gamma_2 \mapsto \gamma_2']$ and where

$(\gamma_1' \Leftarrow^* \gamma_1) \in C \cup \{\gamma_2' \subseteq \gamma_2\}$ and
$\gamma_2 \notin FV(RHS(C), A)$ and $\gamma_2 \neq \gamma_1$ and
$t$, $b$, $LHS(C)$ is monotonic in $\gamma_2$.

Applying Fact 4.14 we get $(S\,\gamma_1' \Leftarrow^* S\,\gamma_1) \in S\,C$ which shows that

$$
A \vdash (S\,C \cup \{S\,\gamma_1' \subseteq S\,\gamma_1\}, S\,t, S\,b) \Leftrightarrow\!\!\!\rightarrow^{\leq 1} (S\,C, S\,t, S\,b)
$$

(if $(S\,\gamma_1' \subseteq S\,\gamma_1) \in S\,C$ we have "=" otherwise "$\Leftrightarrow\!\!\!\rightarrow$"); it is also easy to see that the conditions are fulfilled for applying (shrink) to get

$$
A \vdash (C \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\!\rightarrow (S\,C, S\,t, S\,b)
$$

thus completing the diamond.

**(redund) and (boost)** eliminating $(\gamma_1 \subseteq \gamma_1')$ and boosting $\gamma_2$ into $\gamma_2'$ (with $\gamma_2' \neq \gamma_2$). First notice that it cannot be the case that $(\gamma_1 \subseteq \gamma_1') = (\gamma_2 \subseteq \gamma_2')$, for then we would have $(\gamma_1 \Leftarrow^* \gamma_1') \in C$ (with $C$ the remaining constraints) showing that $\gamma_1 \in LHS(C)$, whereas a side condition for (boost) is that each element in $LHS(C)$ is anti-monotonic in $\gamma_2$.

Now we can proceed as in the case (redund),(shrink).

**(shrink) and (shrink)** shrinking $\gamma_1$ into $\gamma_1'$ and shrinking $\gamma_2$ into $\gamma_2'$ where we can assume that either $\gamma_1' \neq \gamma_2'$ or $\gamma_1 \neq \gamma_2$ as otherwise the claim is trivial. The situation thus is

$$
\begin{aligned}
A \vdash\ & (C \,\dot\cup\, \{\gamma_1' \subseteq \gamma_1\} \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\!\rightarrow \\
& (S_1\,C \cup \{S_1\,\gamma_2' \subseteq S_1\,\gamma_2\}, S_1\,t, S_1\,b) \\
A \vdash\ & (C \,\dot\cup\, \{\gamma_1' \subseteq \gamma_1\} \,\dot\cup\, \{\gamma_2' \subseteq \gamma_2\}, t, b) \Leftrightarrow\!\!\!\rightarrow \\
& (S_2\,C \cup \{S_2\,\gamma_1' \subseteq S_2\,\gamma_1\}, S_2\,t, S_2\,b)
\end{aligned}
$$

where $S_1 = [\gamma_1 \mapsto \gamma_1']$ and $S_2 = [\gamma_2 \mapsto \gamma_2']$. Due to the side conditions for (shrink) we have $\gamma_1 \neq \gamma_2$ and $\gamma_1, \gamma_2 \notin RHS(C)$ so $RHS(S_1\, C) = RHS(C) = RHS(S_2\, C)$ implying $\gamma_1, \gamma_2 \notin RHS(S_1\, C)$ and $\gamma_1, \gamma_2 \notin RHS(S_2\, C)$, thus the "$\cup$" on the right hand sides is really "$\dot\cup$". Our goal then is to find $S_1'$ and $S_2'$ such that

$$\begin{aligned}
&S_2'\, S_1 = S_1'\, S_2 \text{ and}\\
&A \vdash (S_1\, C \,\dot\cup\, \{S_1\, \gamma_2' \subseteq \gamma_2\}, S_1\, t, S_1\, b) \Leftrightarrow\!\!\!\!\rightarrow\\
&\qquad (S_2'\, S_1\, C, S_2'\, S_1\, t, S_2'\, S_1\, b) \text{ and}\\
&A \vdash (S_2\, C \,\dot\cup\, \{S_2\, \gamma_1' \subseteq \gamma_1\}, S_2\, t, S_2\, b) \Leftrightarrow\!\!\!\!\rightarrow\\
&\qquad (S_1'\, S_2\, C, S_1'\, S_2\, t, S_1'\, S_2\, b).
\end{aligned}$$

We naturally define $S_1' = [\gamma_1 \mapsto S_2\, \gamma_1']$ and $S_2' = [\gamma_2 \mapsto S_1\, \gamma_2']$ with the purpose of using (shrink), and our proof obligations are:

$$S_2'\, S_1 = S_1'\, S_2; \tag{4}$$
$$S_2\, \gamma_1' \neq \gamma_1 \text{ and } S_1\, \gamma_2' \neq \gamma_2; \tag{5}$$
$$S_2\, t, S_2\, b, LHS(S_2\, C) \text{ is monotonic in } \gamma_1; \tag{6}$$
$$S_1\, t, S_1\, b, LHS(S_1\, C) \text{ is monotonic in } \gamma_2. \tag{7}$$

Here (4) and (5) amounts to proving that

$$S_2'\, \gamma_1' = S_2\, \gamma_1' \text{ and } S_1\, \gamma_2' = S_1'\, \gamma_2' \text{ and } S_2\, \gamma_1' \neq \gamma_1 \text{ and } S_1\, \gamma_2' \neq \gamma_2 \tag{8}$$

which is trivial if $\gamma_1' \neq \gamma_2$ and $\gamma_2' \neq \gamma_1$. If e.g. $\gamma_1' = \gamma_2$ then we from our assumption about the graph being cycle-free infer that $\gamma_2' \neq \gamma_1$ from which (8) easily follows.

Using Lemma C.1, the claims (6) and (7) are easy consequences of the fact that $t$, $b$ and $LHS(C)$ are monotonic in $\gamma_1$ as well as in $\gamma_2$.

**(boost) and (boost)** where we proceed, *mutatis mutandis*, as in the case (shrink),(shrink).

**(shrink) and (boost)** shrinking $\gamma_1$ into $\gamma_1'$ and boosting $\gamma_2$ into $\gamma_2'$. Let $S_1 = [\gamma_1 \mapsto \gamma_1']$ and $S_2 = [\gamma_2 \mapsto \gamma_2']$. Four cases:

$\underline{\gamma_1 = \gamma_2}$ (to be denoted $\gamma$). Then our assumption about the graph being cycle-free tells us that $\gamma_1' \neq \gamma_2'$, and the situation is

$$A \vdash (C \dot\cup \{\gamma_1' \subseteq \gamma\} \dot\cup \{\gamma \subseteq \gamma_2'\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow (S_1\, C \cup \{\gamma_1' \subseteq \gamma_2'\}, S_1\, t, S_1\, b)$$
$$A \vdash (C \dot\cup \{\gamma_1' \subseteq \gamma\} \dot\cup \{\gamma \subseteq \gamma_2'\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow (S_2\, C \cup \{\gamma_1' \subseteq \gamma_2'\}, S_2\, t, S_2\, b)$$

where (according to the side conditions for (shrink) and (boost)) it holds that $\gamma \notin \mathit{RHS}(C)$ and that $t$, $b$ and each element in $\mathit{LHS}(C)$ is monotonic as well as anti-monotonic in $\gamma$. By Fact 4.18 and using that $C$ is well-formed we infer that $\gamma \notin \mathit{FV}(C, t, b)$, thus the right hand sides of the above two transitions are identical.

$\underline{\gamma_1 = \gamma_2'}$. By the side condition for (shrink) we then have $\gamma_2 = \gamma_1'$. The situation thus is

$$A \vdash (C \dot\cup \{\gamma_2 \subseteq \gamma_1\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow (S_1\, C, S_1\, t, S_1\, b)$$
$$A \vdash (C \dot\cup \{\gamma_2 \subseteq \gamma_1\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow (S_2\, C, S_2\, t, S_2\, b)$$

where the right hand sides are equal modulo renaming.

$\underline{\gamma_2 = \gamma_1'}$. By the side condition for (boost) we then have $\gamma_1 = \gamma_2'$ so we can proceed as in the previous case.

$\underline{\gamma_1 \notin \{\gamma_2, \gamma_2', \gamma_1'\} \text{ and } \gamma_2 \notin \{\gamma_1, \gamma_1', \gamma_2'\}}$ will hold in the remaining case. The situation thus is

$$\begin{aligned} A \vdash\ & (C \dot\cup \{\gamma_1' \subseteq \gamma_1\} \dot\cup \{\gamma_2 \subseteq \gamma_2'\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow \\ & (S_1\, C \cup \{\gamma_2 \subseteq \gamma_2'\}, S_1\, t, S_1\, b) \\ A \vdash\ & (C \dot\cup \{\gamma_1' \subseteq \gamma_1\} \dot\cup \{\gamma_2 \subseteq \gamma_2'\}, t, b) \Leftrightarrow\!\!\!\!\rightarrow \\ & (S_2\, C \cup \{\gamma_1' \subseteq \gamma_1\}, S_2\, t, S_2\, b) \end{aligned}$$

where $\gamma_1 \notin \mathit{FV}(\mathit{RHS}(C), A)$, where $t$ and $b$ and each element in $\mathit{LHS}(C)$ is monotonic in $\gamma_1$, where $\gamma_2 \notin \mathit{FV}(A)$, and where $t$ and $b$ and each element in $\mathit{LHS}(C)$ is anti-monotonic in $\gamma_2$.

As $\gamma_1 \neq \gamma_2'$ it is easy to see (using Lemma C.1) that $\gamma_1 \notin \mathit{FV}(\mathit{RHS}(S_2\, C), A)$ and that $S_2\, t$, $S_2\, b$ and $\mathit{LHS}(S_2\, C)$ is monotonic in $\gamma_1$; and as $\gamma_2 \neq \gamma_1'$ it is easy to see (using Lemma C.1) that $S_1\, t$, $S_1\, b$ and $\mathit{LHS}(S_1\, C)$ is anti-monotonic in $\gamma_2$. Hence the "$\cup$" on the right hand sides is really "$\dot\cup$", and we can apply (boost) and (shrink) to get

$$A \vdash (S_1\, C \dot\cup \{\gamma_2 \subseteq \gamma_2'\}, S_1\, t, S_1\, b) \Leftrightarrow\!\!\!\!\rightarrow (S_2\, S_1\, C, S_2\, S_1\, t, S_2\, S_1\, b)$$
$$A \vdash (S_2\, C \dot\cup \{\gamma_1' \subseteq \gamma_1\}, S_2\, t, S_2\, b) \Leftrightarrow\!\!\!\!\rightarrow (S_1\, S_2\, C, S_1\, S_2\, t, S_1\, S_2\, b)$$

which is as desired since clearly $S_2\, S_1 = S_1\, S_2$. □

## Algorithm $\mathcal{W}$

**Lemma 4.30** Let $C$ be atomic; then

$$C, A \vdash_n e : t \,\&\, b \text{ implies } C, A \vdash_n e : \mathit{GEN}(A, b)(C, t) \,\&\, b.$$

**Proof** Write

$$\{\vec{\gamma}\} = (\mathit{Clos}(FV(t), C)) \setminus (FV(A, b)^{C\downarrow})$$
$$C_0 = C \mid_{\{\vec{\gamma}\}} = \{(g_1 \subseteq g_2) \in C \mid FV(g_1, g_2) \cap \{\vec{\gamma}\} \neq \emptyset\}$$

so that $\mathit{GEN}(A, b)(C, t) = \forall(\vec{\gamma} : C_0).\ t$; this is well-formed by Fact 4.4.

Next let $R$ be a renaming of $\{\vec{\gamma}\}$ into fresh variables. It is immediate that $\forall(\vec{\gamma} : C_0).\ t$ is solvable from $(C \setminus C_0) \cup R\, C_0$ by some $S_0$; simply take $S_0 = R$. Finally note that $\{\vec{\gamma}\} \cap FV((C \setminus C_0) \cup R\, C_0) = \emptyset$ by construction of $C_0$ and $R$, and that $\{\vec{\gamma}\} \cap FV(A, b) = \emptyset$ by construction of $\{\vec{\gamma}\}$.

We then have (using Lemma 2.19 on the assumption) that

$$((C \setminus C_0) \cup R\, C_0) \cup C_0, A \vdash_n e : t \,\&\, b$$

and (gen) gives

$$(C \setminus C_0) \cup R\, C_0, A \vdash_n e : \forall(\vec{\gamma} : C_0).\ t \,\&\, b$$

and finally Lemma 2.18 gives the desired result:

$$(C \setminus C_0) \cup C_0, A \vdash_n e : \forall(\vec{\gamma} : C_0).\ t \,\&\, b.$$

This completes the proof. □

**Theorem 4.31** If $\mathcal{W}(A, e) = (S, t, b, C)$ with $A$ well-formed and $e \in \mathit{EExp}$, then $C, S\,A \vdash_n e : t \,\&\, b$.

**Proof** We proceed by structural induction on $e$; we first prove the result for $\mathcal{W}'$ (using the notation introduced in Fig. 4.1) and then in a joint final case extend the result to $\mathcal{W}$. Notice that by Lemma 4.29, the side condition for applying the induction hypothesis will always be fulfilled.

**The case $e ::= c$ (the case $e ::= x$ is similar).** If $A(c)$ is a type $t$ then the claim is that

$$\emptyset, A \vdash_n c : t \,\&\, \varepsilon$$

but this follows by (con).

Otherwise write $A(c) = \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0$. The claim is that

$$R\,C_0, A \vdash_n c : R\,t_0 \,\&\, \varepsilon$$

where $R$ maps $\vec{\alpha}\vec{\beta}\vec{\rho}$ into fresh variables $\vec{\alpha'}\vec{\beta'}\vec{\rho'}$. But this follows from the inference

$$\begin{array}{ll} R\,C_0, A \vdash c : \forall(\vec{\alpha}\vec{\beta}\vec{\rho} : C_0).\ t_0 \,\&\, \varepsilon & \text{(con)} \\ R\,C_0, A \vdash_n c : R\,t_0 \,\&\, \varepsilon & \text{(ins)} \end{array}$$

where the application of (ins) is justified since $R\,C_0 \vdash R\,C_0$.

**The case $e ::= \texttt{fn}\ x \Rightarrow e_0$.** The induction hypothesis gives

$$C_0, S_0\,(A[x : \alpha]) \vdash_n e_0 : t_0 \,\&\, b_0$$

and using $C = C_0 \cup \{b_0 \subseteq \beta\}$ and $S = S_0$ we get

$$\begin{array}{l} C, S\,(A[x : \alpha]) \vdash_n e_0 : t_0 \,\&\, b_0 \\ C, (S\,A)[x : S\,\alpha] \vdash_n e_0 : t_0 \,\&\, \beta \\ C, S\,A \vdash_n \texttt{fn}\ x \Rightarrow e_0 : S\,\alpha \rightarrow^{\beta} t_0 \,\&\, \varepsilon \end{array}$$

using first Lemma 2.19, then (sub) and finally (abs).

**The case** $e ::= e_1\, e_2$**.** Concerning $e_1$ the induction hypothesis gives

$$C_1, S_1\, A \vdash_n e_1 : t_1 \,\&\, b_1.$$

Using Lemmas 2.18 and 2.19 and then (sub) we get

$$S_2\, C_1, S_2\, S_1\, A \vdash_n e_1 : S_2\, t_1 \,\&\, S_2\, b_1$$
$$C, S\, A \vdash_n e_1 : S_2\, t_1 \,\&\, S_2\, b_1$$
$$C, S\, A \vdash_n e_1 : t_2 \;\rightarrow^{\beta}\; \alpha \,\&\, S_2\, b_1.$$

Turning to $e_2$ the induction hypothesis gives

$$C_2, S_2\, S_1\, A \vdash_n e_2 : t_2 \,\&\, b_2$$

and using Lemma 2.19 we get

$$C, S\, A \vdash_n e_2 : t_2 \,\&\, b_2.$$

Using (app) we get

$$C, S\, A \vdash_n e_1\, e_2 : \alpha \,\&\, S_2\, b_1; b_2; \beta$$

which is the desired result.

**The case** $e ::= e_0\, @^s_n < e_1, \cdots, e_n >$**.** Concerning $e_0$ the induction hypothesis gives

$$C_0, S_0\, A \vdash_n e_0 : t_0 \,\&\, b_0.$$

Using Lemmas 2.18 and 2.19 and then (sub) we get

$$S_n \cdots S_1\, C_0, S\, A \vdash_n e_0 : S_n \cdots S_1\, t_0 \,\&\, S_n \cdots S_1\, b_0$$
$$C, S\, A \vdash_n e_0 : S_n \cdots S_1\, t_0 \,\&\, S_n \cdots S_1\, b_0$$
$$C, S\, A \vdash_n e_0 : S_n \cdots S_2\, t_1 \;\rightarrow\; \cdots t_n \;\rightarrow\; \alpha \,\&\, S_n \cdots S_1\, b_0.$$

For $i \in \{1 \cdots n\}$ the induction hypothesis gives

$$C_i, S_i \cdots S_1\, S_0\, A \vdash_n e_i : t_i \,\&\, b_i$$

and using Lemmas 2.18 and 2.19 we get

$$C, S\, A \vdash_n e_i : S_n \cdots S_{i+1}\, t_i \,\&\, S_n \cdots S_{i+1}\, b_i.$$

Using (sapp) we get

$$C, S\, A \vdash_n e_0 < e_1, \cdots, e_n > : \alpha \,\&\, S_n \cdots S_1\, b_0; \cdots; S_n \cdots S_{i+1}\, b_i; \cdots$$

which is the desired result.

**The case** $e ::= \texttt{let}\ x = e_1\ \texttt{in}\ e_2$**.** Concerning $e_1$ the induction hypothesis gives

$$C_1, S_1\, A \vdash_n e_1 : t_1 \,\&\, b_1$$

and note that by Lemma 4.29 it holds that $C_1$ is atomic. Next let $ts_1 = \mathit{GEN}(S_1\, A, b_1)(C_1, t_1)$ so that Lemmas 4.30, 2.18 and 2.19 give

$$C_1, S_1\, A \vdash_n e_1 : ts_1 \,\&\, b_1$$
$$S_2\, C_1, S\, A \vdash_n e_1 : S_2\, ts_1 \,\&\, S_2\, b_1$$
$$C, S\, A \vdash_n e_1 : S_2\, ts_1 \,\&\, S_2\, b_1.$$

Turning to $e_2$ the induction hypothesis gives

$$C_2, (S_2\, S_1\, A)[x : S_2\, ts_1] \vdash_n e_2 : t_2 \,\&\, b_2$$

and using Lemma 2.19 we get

$$C, (S\, A)[x : S_2\, ts_1] \vdash_n e_2 : t_2 \,\&\, b_2$$

and hence using (let)

$$C, S\, A \vdash_n \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2 \,\&\, S_2\, b_1; b_2$$

and this is the desired result.

**The case $e ::= \texttt{rec}\ f\ x \Rightarrow e_0$.** Concerning $e_0$ the induction hypothesis gives

$$C_0, (S_0\,A)[f : S_0\,\alpha_1 \ \to^{S_0\,\beta}\ S_0\,\alpha_2][x : S_0\,\alpha_1] \vdash_n e_0 : t_0 \,\&\, b_0.$$

Using Lemma 2.19, (sub), (abs) and (rec) we then get

$$C, (S\,A)[f : S\,\alpha_1 \ \to^{S\,\beta}\ S\,\alpha_2][x : S\,\alpha_1] \vdash_n e_0 : t_0 \,\&\, b_0$$
$$C, (S\,A)[f : S\,\alpha_1 \ \to^{S\,\beta}\ S\,\alpha_2][x : S\,\alpha_1] \vdash_n e_0 : S\,\alpha_2 \,\&\, S\,\beta$$
$$C, (S\,A)[f : S\,\alpha_1 \ \to^{S\,\beta}\ S\,\alpha_2] \vdash_n \texttt{fn}\ x \Rightarrow e_0 : S\,\alpha_1 \ \to^{S\,\beta}\ S\,\alpha_2 \,\&\, \varepsilon$$
$$C, S\,A \vdash_n \texttt{rec}\ f\ x \Rightarrow e_0 : S\,\alpha_1 \ \to^{S\,\beta}\ S\,\alpha_2 \,\&\, \varepsilon$$

which is the desired result.

**The case $e ::= \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$.** The induction hypothesis, Lemmas 2.18 and 2.19 and rule (sub) give:

$$C, S\,A \vdash_n e_0 : \texttt{bool} \,\&\, S_2\,S_1\,b_0$$
$$C, S\,A \vdash_n e_1 : \alpha \,\&\, S_2\,b_1$$
$$C, S\,A \vdash_n e_2 : \alpha \,\&\, b_2$$

and rule (if) then gives

$$C, S\,A \vdash_n \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \alpha \,\&\, S_2\,S_1\,b_0; (S_2\,b_1 + b_2)$$

which is the desired result.

**Lifting the result from $\mathcal{W}'$ to $\mathcal{W}$.** We have from the above and Lemma 4.29 that $\mathcal{W}'(A, e) = (S_1, t_1, b_1, C_1)$ with $C_1$ well-formed and that

$$C_1, S_1\,A \vdash_n e : t_1 \,\&\, b_1.$$

Concerning $\mathcal{F}$ we have

$$(S_2, C_2) = \mathcal{F}(C_1)$$

where Lemma 4.10 and Lemma 4.11 ensure that $C_2$ is atomic and that $C_2 \vdash S_2\, C_1$. Using Lemmas 2.18 and 2.19 we get

$$C_2, S_2\, S_1\, A \vdash_n e : S_2\, t_1 \,\&\, S_2\, b_1.$$

Concerning $\mathcal{R}$ we have

$$(C_3, t_3, b_3) = \mathcal{R}(C_2, S_2\, t_1, S_2\, b_1, S_2\, S_1\, A)$$

so by Lemma 4.25 we get

$$C_3, S_2\, S_1\, A \vdash_n e : t_3 \,\&\, b_3$$

which is the desired result. $\square$

# Appendix D

# Proofs of Results Concerning Completeness

## Lazy instance

**Lemma 5.4**

(a) $\leq_C$ is reflexive and transitive.

(b) If $\sigma_1 \leq_C \sigma_2$ and $S$ is a substitution then $S\,\sigma_1 \leq_{S\,C} S\,\sigma_2$.

(c) If $\sigma_1 \leq_C \sigma_2$ and $C' \vdash C$ then $\sigma_1 \leq_{C'} \sigma_2$.

**Proof** Concerning (a) reflexivity of $\leq_C$ is immediate. For transitivity assume that $ts_1 \leq_C ts_2$ and that $ts_2 \leq_C ts_3$; then $C' \vdash C$ and $t' <_{C'} ts_1$ gives first $t' <_{C'} ts_2$ and secondly $t' <_{C'} ts_3$; this shows that $ts_1 \leq_C ts_3$. The entailment property (c) is an immediate consequence of Lemma 2.19 (a), thanks to our "Kripke-semantics". This leaves us with the substitution property (b).

We can, without loss of generality, assume that $\sigma_i = \forall(\vec{\alpha_i}\vec{\beta_i}\vec{\rho_i} : C_i).\, t_i$ and that $\vec{\alpha_i}\vec{\beta_i}\vec{\rho_i}$ does not occur otherwise $(i = 1, 2)$. Then $S\,\sigma_i = \forall(\vec{\alpha_i}\vec{\beta_i}\vec{\rho_i} : S\,C_i).\, S\,t_i$.

Consider $t <_{C'} S\,\sigma_1$ where $C' \vdash S\,C$ and we will prove $t <_{C'} S\,\sigma_2$. Thus we have

$$C' \vdash S_1\,S\,C_1 \tag{1}$$

$$C' \vdash S_1\, S\, t_1 \subseteq t \tag{2}$$

for some $S_1$ with $Dom(S_1) \subseteq \{\vec{\alpha_1}\vec{\beta_1}\vec{\rho_1}\}$. Clearly $t_1 <_{C \cup C_1} \sigma_1$ so using $\sigma_1 \leq_C \sigma_2$ we get (again thanks to our "Kripke-semantics") $t_1 <_{C \cup C_1} \sigma_2$. This means that

$$C \cup C_1 \vdash S_0\, C_2 \tag{3}$$
$$C \cup C_1 \vdash S_0\, t_2 \subseteq t_1 \tag{4}$$

for some $S_0$ with $Dom(S_0) \subseteq \{\vec{\alpha_2}\vec{\beta_2}\vec{\rho_2}\}$. Since $\{\vec{\alpha_1}\vec{\beta_1}\vec{\rho_1}\}$ does not occur in $C$ nor in $Dom(S) \cup Ran(S)$ we have $S_1\, S\, C = S\, C$ so from $C' \vdash S\, C$ we get $C' \vdash S_1\, S\, C$. Using (1) we therefore have $C' \vdash S_1\, S\, (C \cup C_1)$ and Lemmas 2.19 and 2.18 applied to (3) and (4) give

$$C' \vdash S_1\, S\, S_0\, C_2 \tag{5}$$
$$C' \vdash S_1\, S\, S_0\, t_2 \subseteq S_1\, S\, t_1.$$

Using (2) the latter yields

$$C' \vdash S_1\, S\, S_0\, t_2 \subseteq t. \tag{6}$$

Now define $S_2 = [\vec{\alpha_2}\vec{\beta_2}\vec{\rho_2} \mapsto S_1\, S\, S_0(\vec{\alpha_2}\vec{\beta_2}\vec{\rho_2})]$. Below we show that

$$S_2\, S\, \gamma = S_1\, S\, S_0\, \gamma \text{ for } \gamma \in FV(t_2, C_2) \tag{7}$$

so (5) and (6) can be rewritten as

$$C' \vdash S_2\, S\, C_2$$
$$C' \vdash S_2\, S\, t_2 \subseteq t$$

showing that $t <_{C'} S\, \sigma_2$.

To prove (7) assume first that $\gamma \in FV(t_2, C_2) \cap \{\vec{\alpha_2}\vec{\beta_2}\vec{\rho_2}\}$. Then

$$\begin{aligned} S_2\, S\, \gamma &= S_2\, \gamma && \text{since } \gamma \notin Dom(S) \\ &= S_1\, S\, S_0\, \gamma && \text{by definition of } S_2 \end{aligned}$$

Next assume $\gamma \in FV(t_2, C_2) \setminus \{\vec{\alpha_2}\vec{\beta_2}\vec{\rho_2}\}$. Then

$$\begin{array}{rcll} S_2\,S\,\gamma &=& S\,\gamma & \text{since } Dom(S_2) \cap FV(S\,\gamma) = \emptyset \\ &=& S_1\,S\,\gamma & \text{since } Dom(S_1) \cap FV(S\,\gamma) = \emptyset \text{ as } \gamma \notin \{\vec{\alpha_1}\vec{\beta_1}\vec{\rho_1}\} \\ &=& S_1\,S\,S_0\,\gamma & \text{since } \gamma \notin Dom(S_0) \end{array}$$

This completes the proof. □

**Lemma 5.7**

(a) $\preceq^{\mathrm{Id}}$ is reflexive and transitive.

(b) If $jdg_1 \preceq^{\mathrm{Id}} jdg_2$ and $S$ is a substitution then $S\,jdg_1 \preceq^{\mathrm{Id}} S\,jdg_2$.

(c) If $C_1, A_1 \mid e : \sigma_1 \,\&\, b_1 \preceq^{\mathrm{Id}} jdg_2$ and $C_0 \vdash C_1$ then $C_0, A_1 \mid e : \sigma_1 \,\&\, b_1 \preceq^{\mathrm{Id}} jdg_2$.

**Proof** Concerning (a), reflexivity of $\preceq^{Id}$ is immediate. For transitivity assume that $jdg_1 \preceq^{Id} jdg_2$ and that $jdg_2 \preceq^{Id} jdg_3$ and that $jdg_i = C_i, A_i \mid e : \sigma_i \& b_i$; then $C_1 \vdash C_2$ and $C_2 \vdash C_3$ give $C_1 \vdash C_3$, $A_2 \leq_{C_1} A_1$ and $A_3 \leq_{C_2} A_2$ give $A_3 \leq_{C_1} A_1$ (by Lemma 5.4), $\sigma_1 \leq_{C_1} \sigma_2$ and $\sigma_2 \leq_{C_2} \sigma_3$ give $\sigma_1 \leq_{C_1} \sigma_3$ (by Lemma 5.4), and $C_1 \vdash b_2 \subseteq b_1$ and $C_2 \vdash b_3 \subseteq b_2$ give $C_1 \vdash b_3 \subseteq b_1$; this shows that $jdg_1 \preceq^{Id} jdg_3$.

For the substitution property (b) assume that

$$C_1, A_1 \mid e : \sigma_1 \;\&\; b_1 \preceq^{Id} C_2, A_2 \mid e : \sigma_2 \;\&\; b_2$$

and show

$$S\,C_1, S\,A_1 \mid e : S\,\sigma_1 \;\&\; S\,b_1 \preceq^{Id} S\,C_2, S\,A_2 \mid e : S\,\sigma_2 \;\&\; S\,b_2.$$

Now note that by Lemmas 2.18 and 5.4 we have

$C_1 \vdash C_2$ implies $S\,C_1 \vdash S\,C_2$

$A_2 \leq_{C_1} A_1$ implies $S\,A_2 \leq_{S\,C_1} S\,A_1$

$\sigma_1 \leq_{C_1} \sigma_2$ implies $S\,\sigma_1 \leq_{S\,C_1} S\,\sigma_2$

$C_1 \vdash b_2 \subseteq b_1$ implies $S\,C_1 \vdash S\,b_2 \subseteq S\,b_1$.

For the entailment property (c) assume that

$$C_1, A_1 \mid e : \sigma_1 \;\&\; b_1 \preceq^{Id} C_2, A_2 \mid e : \sigma_2 \;\&\; b_2 \text{ and } C_0 \vdash C_1$$

and show

$$C_0, A_1 \mid e : \sigma_1 \;\&\; b_1 \preceq^{Id} C_2, A_2 \mid e : \sigma_2 \;\&\; b_2.$$

Now note that by Lemma 2.19 and Lemma 5.4 we have

$$C_1 \vdash C_2 \text{ implies } C_0 \vdash C_2$$
$$A_2 \leq_{C_1} A_1 \text{ implies } A_2 \leq_{C_0} A_1$$
$$\sigma_1 \leq_{C_1} \sigma_2 \text{ implies } \sigma_1 \leq_{C_0} \sigma_2$$
$$C_1 \vdash b_2 \subseteq b_1 \text{ implies } C_0 \vdash b_2 \subseteq b_1.$$

This completes the proof. □

**Lemma 5.8** If $C^*, A^* \mid e : t^* \,\&\, b^* \preceq^S C, A \mid e : t \,\&\, b$ then $C^*, A^* \mid e : t^* \,\&\, b^* \preceq^S C, A \mid e : \mathit{GEN}(A, b)(C, t) \,\&\, b$.

**Proof** Assume the hypothesis; then we have

$$C^* \vdash S\, C$$
$$C^* \vdash S\, t \subseteq t^*$$

and by Fact 5.3 it suffices to prove

$$t^* <_{C^*} S\,(GEN(A, b)(C, t)).$$

For this write

$$GEN(A, b)(C, t) = \forall(G : C \mid_G).\ t$$

and note that since $S\,R\,(C \mid_G) = S\,(R\,C \mid_{(R\,G)}) = (S\,R\,C) \mid_{(R\,G)}$ we have

$$S\,(GEN(A, b)(C, t)) = \forall(R\,G : (S\,R\,C) \mid_{(R\,G)}).\ S\,R\,t$$

for a renaming $R$ that maps $G$ into fresh variables. Next define $S'$ by

$$S'\,\gamma = \begin{cases} S\,(R^{-1}\,\gamma) & \text{if } \gamma \in R\,G \\ \gamma & \text{otherwise} \end{cases}$$

and note that $S'\,S\,R = S$ on $FV(t, C)$. Therefore we have the desired judgements

$$C^* \vdash S'\,((S\,R\,C)\,|_{(R\,G)})$$
$$C^* \vdash S'\,(S\,R\,t) \subseteq t^*.$$

This completes the proof. □

# Algorithm $\mathcal{F}$

First a fact which in effect says that we do not have infinite types:

**Fact D.1** If $t \approx sh[\cdots t \cdots, \vec{\beta}, \vec{\rho}]$ then $sh$ is $[\,]$.

**Proof** There exists unique decompositions such that $t = sh_1[\vec{\alpha_1}, \vec{\beta_1}, \vec{\rho_1}]$ and $sh[\cdots t \cdots, \vec{\beta}, \vec{\rho}] = sh_2[\vec{\alpha_2}, \vec{\beta_2}, \vec{\rho_2}]$ and by definition of $t \approx sh[\cdots t \cdots, \vec{\beta}, \vec{\rho}]$ we have $sh_1 = sh_2$. Clearly $sh_2$ must be of the form $sh[\cdots sh_1 \cdots]$ and (say by counting symbols) $sh_1 = sh_2$ is only possible if $sh$ is $[\,]$. □

**Lemma D.2** If $R$ is a matching substitution for the well-formed constraint set $C$, $\alpha' \sim \alpha''$ implies $R\,\alpha' \approx R\,\alpha''$, and $(S, C, \sim) \Leftrightarrow\!\!\!\rightarrow (S', C', \sim')$; then there exists $R'$ and $T$ such that $S' = T\,S$, $R \underset{NF}{=} R'\,T$, $R'$ is a matching substitution for $C'$, and $\alpha' \sim' \alpha''$ implies $R'\,\alpha' \approx R'\,\alpha''$ (where $NF$ is the complement of the set $F$ of fresh variables generated).

If $C^* \vdash R\,C$ with $C^*$ atomic, then (by Fact 5.11) $R$ is a matching substitution for $C$, and the substitution $R'$ mentioned above can be chosen such that $C^* \vdash R'\,C'$.

**Proof** We perform case analysis on Figure 4.3.

**The case (dc).** Here $S' = S$ and $\sim'=\sim$ and $F = \emptyset$; we choose $T = \mathrm{Id}$ and $R' = R$. Our task is to show that $R$ is a matching substitution for $C'$ and that $C^* \vdash R\,C$ implies $C^* \vdash R\,C'$. But the former follows from the remark after Fact 5.10; and the latter is trivial using the rules labelled (bw).

**The cases (mr) and (ml)** are rather similar and we only consider (ml) in detail. Here $\mathcal{M}(\alpha, t, \sim, T, \sim')$ holds and $C' = T\,C$. Considering the definition of $\mathcal{M}$ in Figure 4.4, our assumptions ensure that $R\,\alpha_i \approx R\,\alpha$ (for all $i \in \{1 \cdots n\}$) and $R\,\alpha \approx R\,t$. Since $R\,t = sh[R\,\vec{\alpha_0}\,, R\,\vec{\beta_0}\,, R\,\vec{\rho_0}\,]$ this gives

$$R\,\alpha_i \approx sh[R\,\vec{\alpha_0}\,, R\,\vec{\beta_0}\,, R\,\vec{\rho_0}\,] \text{ (for all } i \in \{1 \cdots n\}).$$

It is then easy to see that we can find $\vec{t_i}$ and $\vec{\beta'_i}$ and $\vec{\rho'_i}$ such that $R\,\alpha_i = sh[\vec{t_i}\,, \vec{\beta'_i}\,, \vec{\rho'_i}\,]$ (for all $i \in \{1 \cdots n\}$). We now define $R'$ by

$$R'\,\gamma = \begin{cases} t_{ij} & \text{if } \gamma = \alpha_{ij} \ (i > 0) \\ \beta'_{ik} & \text{if } \gamma = \beta_{ik} \ (i > 0) \\ \rho'_{il} & \text{if } \gamma = \rho_{il} \ (i > 0) \\ R\,\gamma & \text{otherwise} \end{cases}$$

and it follows that $R'\,T \underset{NF}{=} R$ since $F = \{\alpha_{ij}, \beta_{ik}, \rho_{il} \mid i, j, k, l > 0\}$.
Since $R\,C = R'\,T\,C = R'\,C'$ the remaining claims follow, except to show that $\alpha' \sim' \alpha''$ implies $R'\,\alpha' \approx R'\,\alpha''$. Since $\approx$ is an equivalence relation it suffices to consider the two base cases in the construction of $\sim'$. One is when $\alpha' \sim \alpha''$ and $\{\alpha', \alpha''\} \cap \{\alpha_1, \cdots, \alpha_n\} = \emptyset$; here $R\,\alpha' \approx R\,\alpha''$ so $R'\,\alpha' = R'\,T\,\alpha' = R\,\alpha' \approx R\,\alpha'' = R'\,T\,\alpha'' = R'\,\alpha''$. The other is when $\alpha_{0j} \sim' \alpha_{ij}$ (for $i > 0$). We have

$$\begin{array}{rl} & sh[R'\,\vec{\alpha_0}\,, R'\,\vec{\beta_0}\,, R'\,\vec{\rho_0}\,] = sh[R\,\vec{\alpha_0}\,, R\,\vec{\beta_0}\,, R\,\vec{\rho_0}\,] \\ \approx & R\,\alpha_i = sh[\vec{t_i}\,, \vec{\beta'_i}\,, \vec{\rho'_i}\,] = sh[R'\,\vec{\alpha_i}\,, R'\,\vec{\beta_i}\,, R'\,\vec{\rho_i}\,] \end{array}$$

and from the remark after Fact 5.10 we conclude $R'\,\alpha_{0j} \approx R'\,\alpha_{ij}$ (for $i > 0$). This completes the proof of Lemma D.2. $\Box$

**Lemma D.3** Suppose $R$ is a matching substitution for the well-formed constraint set $C$, $\alpha' \sim \alpha''$ implies $R\,\alpha' \approx R\,\alpha''$, and that $(S, C, \sim) \not\Rightarrow$. Then $C$ is atomic.

**Proof** Our task is to show that $C$ cannot contain constraints of the form $t_1 \subseteq t_2$, with $t_1$ and $t_2$ non-variables, and that $C$ cannot contain constraints of the form $\alpha \subseteq t$ or $t \subseteq \alpha$, with $t$ a non-variable.

For the former claim observe that $R\,t_1 \approx R\,t_2$ forces $t_1$ and $t_2$ to have the same top-level type constructor and this contradicts our assumption that $(S, C, \sim) \not\Leftrightarrow\!\!\!\rightarrow$.

For the latter claim it suffices to demonstrate that if $R\,\alpha \approx R\,t$ with $t$ a non-variable type then the "call" $\mathcal{M}(\alpha, t, \sim)$ succeeds, that is there exists $R'$ and $\sim'$ such that $\mathcal{M}(\alpha, t, \sim, R', \sim')$ holds. Let $sh[\vec{\alpha}\,, \vec{\beta}\,, \vec{\rho}]$ be the unique decomposition of $t$. Assume for the sake of arriving at a contradiction that $\alpha'$ is such that $\alpha' \sim \alpha$ and $\alpha' \in FV(t)$ (i.e. $\alpha' \in \vec{\alpha}$); then $R\,\alpha' \approx R\,\alpha \approx R\,t = sh[R\,\vec{\alpha}\,, R\,\vec{\beta}\,, R\,\vec{\rho}] = sh[\cdots R\,\alpha' \cdots, R\,\vec{\beta}\,, R\,\vec{\rho}]$ so by Fact D.1 we infer $sh = [\,]$ and hence $t$ is a variable, yielding the desired contradiction. □

**Lemma 5.12** Suppose that $C$ is well-formed and that $R$ is a matching substitution for $C$. Then $\mathcal{F}(C)$ will always succeed, and whenever $\mathcal{F}(C) = (S', C')$ there exists $R'$ such that $R'$ is a matching substitution for $C'$ and $R \underset{NF(C)}{=\!=} R'\,S'$, where $NF(C)$ is the complement of the set $F(C)$ of fresh variables generated in the call $\mathcal{F}(C)$.

If $C$ is well-formed and $C^* \vdash R\,C$ with $C^*$ atomic, then (by Fact 5.11) $R$ is a matching substitution for $C$, and whenever $\mathcal{F}(C)$ succeeds with result $(S', C')$ the substitution $R'$ mentioned in the first part of the lemma can be chosen such that $C^* \vdash R'\,C'$.

**Proof** We know by Lemma 4.10 that $\mathcal{F}$ will terminate. It will be sufficient to prove that for all sequences

$$(\mathrm{Id}, C, \mathrm{Eq}_C) = (S_0, C_0, \sim_0) \Leftrightarrow\!\!\!\rightarrow^* (S_i, C_i, \sim_i)$$

(where by Fact 4.9 each $C_i$ is well-formed) there exists $R_i$ such that

$$R_i \text{ is a matching substitution for } C_i,\ \alpha' \sim_i \alpha'' \text{ implies } R_i\,\alpha' \approx R_i\,\alpha'',$$
$$R \underset{NF_i}{=\!=} R_i\,S_i, \text{ and } C^* \vdash R\,C \text{ implies } C^* \vdash R_i\,C_i$$

(where $NF_i$ is the complement of the set $F_i$ of fresh variables generated in the first $i$ steps) for then Lemma D.3 will ensure that if $(S_i, C_i, \sim_i) \not\Leftrightarrow\!\!\!\rightarrow$ then

$C_i$ is atomic and hence $\mathcal{F}$ will succeed.

The claim above will be proved by induction in $i$, where the base case is immediate when we take $R_0 = R$.

For the inductive step we simply make use of Lemma D.2 and construct $R_{i+1}$ as the $R'$ guaranteed by that lemma; in particular notice that if $\gamma \notin F_{i+1}$ then $\gamma \notin F_i$ and neither $\gamma$ nor $S_i\,\gamma$ are fresh in the induction step, so $R_{i+1}\,S_{i+1}\,\gamma = R_{i+1}\,T\,S_i\,\gamma = R_i\,S_i\,\gamma = R\,\gamma$. $\square$

# Completeness of Algorithm $\mathcal{W}$

**Theorem 5.18**

If $C^*, A^* \vdash_n^{at} e : \sigma^* \,\&\, b^*$ and
$C^*$ is atomic and
$A^* \leq_{C^*} S''\,A$ with $A$ well-formed
then there exists $S$, $t$, $b$, $C$, and $S'$ such that
$\mathcal{W}(A, e) = (S, t, b, C)$
$S'' \overline{\overline{\scriptstyle NF(A,e)}}\; S'\,S$
$C^*, S''\,A \mid e : \sigma^* \,\&\, b^* \preceq^{S'} C, S\,A \mid e : GEN(S\,A, b)(C, t) \,\&\, b$

**Proof** We assume that

$$C^*, A^* \vdash_n^{at} e : \sigma^* \,\&\, b^* \tag{8}$$

$$C^* \text{ is atomic} \tag{9}$$

$$A^* \leq_{C^*} S''\,A \text{ with } A \text{ well-formed} \tag{10}$$

(in particular, $A^*(x)/A^*(c)$ is a type scheme iff $A(x)/A(c)$ is a type scheme) and proceed by induction on the structure of the normalised proof tree of (8), cf. Definition 2.22. In all cases we must find $S$, $t$, $b$, $C$ and $S'$ such that

$$\mathcal{W}(A, e) = (S, t, b, C) \tag{11}$$

$$S'' \overline{\overline{\scriptstyle NF(A,e)}}\; S'\,S \tag{12}$$

$$C^*, S''\,A \mid e : \sigma^* \,\&\, b^* \preceq^{S'} C, S\,A \mid e : GEN(S\,A, b)(C, t) \,\&\, b. \tag{13}$$

In the case of a T-normalised proof tree (where $\sigma^*$ is a type) we first prove that there exist $S$, $t$, $b$, $C$, $S'$ such that

$$\mathcal{W}'(A, e) = (S, t, b, C) \tag{14}$$

$$S'' \underset{NF'(A,e)}{=\!=\!=} S'\, S \tag{15}$$

$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \;\preceq^{S'}\; C, S\, A \mid e \,:\, t \,\&\, b \tag{16}$$

(where $NF'(A, e)$ is the complement of the set of freshly generated variables during the call $\mathcal{W}'(A, e)$) and then in a common final case we lift the reasoning and find (another) $S$, $t$, $b$, $C$, $S'$ such that (11), (12) and (16) holds (this shall frequently be used in the inductive proof). By Lemma 5.8 we immediately get (13) from (16) since $\sigma^*$ is a type.

In the case of a TS-normalised proof tree (where $\sigma^*$ is a type scheme) we directly prove (11), (12), and (13).

**The case (id).** (The case (con) is similar.) The proof tree of (8) must have the form

$$\frac{}{C^*, A^* \vdash x \,:\, t^* \,\&\, \varepsilon}\;\text{(id)}$$

where $t^* = A^*(x)$. Now let

$$S = \mathrm{Id},\;\; t = A(x),\;\; b = \varepsilon,\;\; C = \emptyset,\;\; S' = S''$$

which establishes (14) as well as (15). The claim (16) amounts to

$$C^*, S''\, A \mid e \,:\, A^*(x) \,\&\, \varepsilon \;\preceq^{S''}\; \emptyset, A \mid e \,:\, A(x) \,\&\, \varepsilon$$

which is a consequence of (10).

**The case (id)(ins).** (The case (con)(ins) is similar.) The proof tree of (8) must have the form

$$\frac{\dfrac{}{C^*, A^* \vdash x \,:\, \forall(G_0^* : C_0^*).\; t_0^* \,\&\, \varepsilon}\;\text{(id)}}{C^*, A^* \vdash x \,:\, S_0^*\, t_0^* \,\&\, \varepsilon}\;\text{(ins)}$$

where $A^*(x) = \forall(G_0^* : C_0^*).\ t_0^*$, $Dom(S_0^*) \subseteq G_0^*$, and $C^* \vdash S_0^*\, C_0^*$. Next let $A(x) = \forall(G_0 : C_0).\ t_0$, let $R$ be the renaming of $G_0$ performed by the algorithm (in $INST$), and let $R'$ be a renaming of $G_0$ into variables not in $Dom(S'') \cup Ran(S'') \cup FV(t_0, C_0)$. Now set

$$S = \mathrm{Id},\ t = R\, t_0,\ b = \varepsilon,\ C = R\, C_0$$

and note that this establishes (14). From (10) we have $\forall(G_0^* : C_0^*).\ t_0^* \leq_{C^*} S''\,(\forall(G_0 : C_0).\ t_0)$ and since $S_0^*\, t_0^* <_{C^*} \forall(G_0^* : C_0^*).\ t_0^*$ is ensured by our assumptions we have

$$S_0^*\, t_0^* <_{C^*} S''\,(\forall(G_0 : C_0).\ t_0) = \forall(R'\, G_0 : S''\, R'\, C_0).\ S''\, R'\, t_0.$$

Hence there exists $S_0'$ with $Dom(S_0') \subseteq R'\, G_0$ such that

$$C^* \vdash S_0'\, S''\, R'\, C_0$$
$$C^* \vdash S_0'\, S''\, R'\, t_0 \subseteq S_0^*\, t_0^*.$$

Now define $S'$ by

$$S'\,\gamma = \begin{cases} S_0'\, R'\,\gamma' & \text{if } \gamma = R\,\gamma' \text{ with } \gamma' \in G_0 \\ S''\,\gamma & \text{otherwise} \end{cases}$$

and note that this establishes (15). Next note that

$$S'\, R = S_0'\, S''\, R' \text{ on } FV(t_0, C_0)$$

so that we already have

$$C^* \vdash S'\, C$$
$$C^* \vdash S'\, t \subseteq S_0^*\, t_0^*$$

and by Fact 5.3 this then establishes (16).

**The case (abs).** The proof tree in (8) must have the form

$$\frac{\dfrac{\vdots}{C^*, A^*[x : t_1^*] \vdash_n^{at} e_0 : t_0^* \,\&\, \beta^*}}{C^*, A^* \vdash_n^{at} \texttt{fn}\ x \Rightarrow e_0 : t_1^* \to^{\beta^*} t_0^* \,\&\, \varepsilon} \quad \text{(abs)}$$

With $\alpha$ the fresh variable chosen by the algorithm and with $S''_\alpha = S''[\alpha \mapsto t_1^*]$ we have from (10) that

$$A^*[x : t_1^*] \leq_{C^*} S''_\alpha\,(A[x : \alpha]).$$

This enables us to apply the induction hypothesis which (since (11), (12), (16) holds) gives us $S_0$, $t_0$, $b_0$, $C_0$ and $S'_0$ such that

$$\begin{array}{l}
\mathcal{W}(A[x : \alpha], e_0) = (S_0, t_0, b_0, C_0) \\
S''_\alpha \overline{\overline{\quad}}_{NF(A[x:\alpha], e_0)}\, S'_0\, S_0 \\
C^*, S''_\alpha\,(A[x : \alpha]) \mid e_0 : t_0^* \,\&\, \beta^* \preceq^{S'_0} \\
C_0, (S_0\, A)[x : S_0\, \alpha] \mid e_0 : t_0 \,\&\, b_0.
\end{array} \qquad (17)$$

To establish (14) we now set

$$S = S_0, t = S_0\, \alpha \to^\beta t_0, b = \varepsilon, C = C_0 \cup \{b_0 \subseteq \beta\}$$

for $\beta$ the fresh variable chosen by the algorithm. Setting

$$S' = S'_0[\beta \mapsto \beta^*]$$

establishes (15) since for $\gamma \in NF'(A, \texttt{fn}\ x \Rightarrow e_0)$ we have $S'\, S\, \gamma = S'_0\, S_0\, \gamma = S''_\alpha\, \gamma = S''\, \gamma$; in addition it holds that $S'\, S\, \alpha = S'_0\, S_0\, \alpha = S''_\alpha\, \alpha = t_1^*$. Our final task is to establish (16), i.e. to ensure that

$$C^*, S''\, A \mid e : t_1^* \to^{\beta^*} t_0^* \,\&\, \varepsilon \preceq^{S'} C, S\, A \mid e : t \,\&\, \varepsilon$$

but using the previous results, in particular (17), this follows from the following observations:

$$C^* \vdash S'_0\, C_0 = S'\, C_0$$
$$C^* \vdash S'\, b_0 = S'_0\, b_0 \subseteq \beta^* = S'\, \beta$$
$$C^* \vdash S'\, t = t_1^* \rightarrow^{\beta^*} S'_0\, t_0 \subseteq t_1^* \rightarrow^{\beta^*} t_0^*.$$

**The case (app).** The proof tree in (8) must have the form

$$\dfrac{\dfrac{\vdots}{C^*, A^* \vdash_n^{at} e_1 : t_2^* \rightarrow^{\beta^*} t^* \,\&\, b_1^*} \qquad \dfrac{\vdots}{C^*, A^* \vdash_n^{at} e_2 : t_2^* \,\&\, b_2^*}}{C^*, A^* \vdash_n^{at} e_1\, e_2 : t^* \,\&\, b_1^*; b_2^*; \beta^*} \quad \text{(app)}$$

Since $A^* \leq_{C^*} S''\, A$ the induction hypothesis gives $S_1, t_1, b_1, C_1$ and $S'_1$ such that

$$\mathcal{W}(A, e_1) = (S_1, t_1, b_1, C_1)$$
$$S'' \overline{\overline{\ \ }}_{NF(A, e_1)}\ S'_1\, S_1$$
$$C^*, S''\, A \mid e_1 : t_2^* \rightarrow^{\beta^*} t^* \,\&\, b_1^* \preceq^{S'_1} C_1, S_1\, A \mid e_1 : t_1 \,\&\, b_1. \tag{18}$$

We thus have $A^* \leq_{C^*} S'_1\, S_1\, A$ and as $S_1\, A$ is well-formed we can apply the induction hypothesis once more to find $S_2, t_2, b_2, C_2$ and $S'_2$ such that

$$\mathcal{W}(S_1\, A, e_2) = (S_2, t_2, b_2, C_2)$$
$$S'_1 \overline{\overline{\ \ }}_{NF(S_1\, A, e_2)}\ S'_2\, S_2 \tag{19}$$
$$C^*, S'_1\, S_1\, A \mid e_2 : t_2^* \,\&\, b_2^* \preceq^{S'_2} C_2, S_2\, S_1\, A \mid e_2 : t_2 \,\&\, b_2. \tag{20}$$

Given (19) we may replace $S'_1$ in (18) by $S'_2\, S_2$ so that we have

$$C^*, S''\, A \mid e_1 : t_2^* \rightarrow^{\beta^*} t^* \,\&\, b_1^* \preceq^{S'_2}$$
$$S_2\, C_1, S_2\, S_1\, A \mid e_1 : S_2\, t_1 \,\&\, S_2\, b_1. \tag{21}$$

To establish (14) we now set

$$S = S_2\, S_1,\ t = \alpha,\ b = S_2\, b_1; b_2; \beta,$$
$$C = S_2\, C_1 \cup C_2 \cup \{S_2\, t_1 \subseteq t_2 \rightarrow^{\beta} \alpha\}$$

for $\alpha$ and $\beta$ the fresh variables chosen by the algorithm. Setting

$$S' = S_2'[\alpha \mapsto t^*,\ \beta \mapsto \beta^*]$$

establishes (15) since for $\gamma$ in $NF'(A, e_1\ e_2)$ we have $FV(S\,\gamma) \cap \{\alpha, \beta\} = \emptyset$ and $FV(S_1\,\gamma) \subseteq NF(S_1\,A, e_2)$ and therefore $S'\,S\,\gamma = S_2'\,S_2\,S_1\,\gamma = S_1'\,S_1\,\gamma = S''\,\gamma$.

Our final task is to establish (16), i.e. to ensure that

$$C^*, S''\,A \mid e\ :\ t^*\,\&\,b_1^*; b_2^*; \beta^*\ \preceq^{S'}\ C, S\,A \mid e\ :\ \alpha\,\&\,S_2\,b_1; b_2; \beta$$

but this is an immediate consequence of (20) and (21) where $S_2'$ can be replaced by $S'$, in particular we employ that $C^* \vdash S'\,t_2 \subseteq t_2^*$ and hence

$$C^* \vdash S'\,S_2\,t_1 \subseteq t_2^* \to^{\beta^*} t^* \subseteq S'\,(t_2 \to^{\beta} \alpha).$$

**The case (sapp).** Is quite similar to (app).

**The case (rec).** The normalised proof tree in (8) must have the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{C^*, A^*[f : t^*][x : t_1^*] \vdash_n^{at} e_0\ :\ t_2^*\,\&\,\beta^*}}{C^*, A^*[f : t^*] \vdash_n^{at} \mathtt{fn}\ x \Rightarrow e_0\ :\ t_1^* \to^{\beta^*} t_2^*\,\&\,\varepsilon}\ (\text{abs})}{C^*, A^*[f : t^*] \vdash_n^{at} \mathtt{fn}\ x \Rightarrow e_0\ :\ t^*\,\&\,b^*}\ (\text{sub})^*}{C^*, A^* \vdash_n^{at} \mathtt{rec}\ f\ x \Rightarrow e_0\ :\ t^*\,\&\,b^*}\ (\text{rec})$$

where

$$C^* \vdash t_1^* \to^{\beta^*} t_2^* \subseteq t^* \text{ and } C^* \vdash \varepsilon \subseteq b^*. \tag{22}$$

Next define

$$S_0'' = S''[\alpha_1 \mapsto t_1^*,\ \alpha_2 \mapsto t_2^*][\beta \mapsto \beta^*]$$

with $\alpha_1$, $\alpha_2$ and $\beta$ the fresh variables chosen by the algorithm; then we from (10) infer that

$$A^*[f : t^*][x : t_1^*] \leq_{C^*} S_0''\,(A[f : \alpha_1 \;\to^\beta\; \alpha_2][x : \alpha_1])$$

and hence we can use the induction hypothesis to find $S_0, t_0, b_0, C_0$ and $S_0'$ such that

$$\begin{array}{l}
\mathcal{W}(A[f : \alpha_1 \;\to^\beta\; \alpha_2][x : \alpha_1], e_0) = (S_0, t_0, b_0, C_0) \\
S_0'' \;\overline{\overline{\scriptstyle NF(A[f\,:\,\cdots][x\,:\,\alpha_1],\,e)}}\; S_0'\, S_0 \\
C^*, S_0''\,(A[f : \alpha_1 \;\to^\beta\; \alpha_2][x : \alpha_1]) \mid e_0 \;:\; t_2^* \,\&\, \beta^* \;\preceq^{S_0'} \\
C_0, S_0\,(A[f : \alpha_1 \;\to^\beta\; \alpha_2][x : \alpha_1]) \mid e_0 \;:\; t_0 \,\&\, b_0.
\end{array} \tag{23}$$

To establish (14) we now set

$$S = S_0,\; t = S_0\,(\alpha_1 \;\to^\beta\; \alpha_2),\; b = \varepsilon,\; C = C_0 \cup \{b_0 \subseteq S_0\,\beta, t_0 \subseteq S_0\,\alpha_2\}$$

and we clearly establish (15) by setting

$$S' = S_0'.$$

Our final task is to establish (16), i.e. to ensure that

$$C^*, S''\,A \mid e \;:\; t^* \,\&\, b^* \;\preceq^{S_0'}\; C, S_0\,A \mid e \;:\; S_0\,(\alpha_1 \;\to^\beta\; \alpha_2) \,\&\, \varepsilon$$

but this follows from the following observations (where we use (23) and (22)):

$$\begin{array}{l}
C^* \vdash S_0'\,b_0 \subseteq \beta^* = S_0''\,\beta = S_0'\,S_0\,\beta \\
C^* \vdash S_0'\,t_0 \subseteq t_2^* = S_0''\,\alpha_2 = S_0'\,S_0\,\alpha_2 \\
C^* \vdash S_0'\,S_0\,(\alpha_1 \;\to^\beta\; \alpha_2) = S_0''\,(\alpha_1 \;\to^\beta\; \alpha_2) = t_1^* \;\to^{\beta^*}\; t_2^* \subseteq t^* \\
C^* \vdash S_0'\,\varepsilon \subseteq b^*.
\end{array}$$

**The case (if)**. The immediate premises of the inference (8) must have the form

$$\begin{array}{l}
C^*, A^* \vdash_n^{at} e_0 \;:\; \texttt{bool} \,\&\, b_0^* \\
C^*, A^* \vdash_n^{at} e_1 \;:\; t^* \,\&\, b_1^* \\
C^*, A^* \vdash_n^{at} e_2 \;:\; t^* \,\&\, b_2^*.
\end{array}$$

Since $A^* \leq_{C^*} S'' A$ the induction hypothesis gives

$$\mathcal{W}(A, e_0) = (S_0, t_0, b_0, C_0)$$
$$S'' \overline{\overline{\scriptstyle NF(A,e_0)}}\ S'_0 S_0$$
$$C^*, S'' A \mid e_0 : \texttt{bool} \,\&\, b^*_0 \preceq^{S'_0} C_0, S_0 A \mid e_0 : t_0 \,\&\, b_0.$$

We thus have $A^* \leq_{C^*} S'_0 S_0 A$ and as $S_0 A$ is well-formed we can apply the induction hypothesis once more giving

$$\mathcal{W}(S_0 A, e_1) = (S_1, t_1, b_1, C_1)$$
$$S'_0 \overline{\overline{\scriptstyle NF(S_0 A,e_1)}}\ S'_1 S_1$$
$$C^*, S'_0 S_0 A \mid e_1 : t^* \,\&\, b^*_1 \preceq^{S'_1} C_1, S_1 S_0 A \mid e_1 : t_1 \,\&\, b_1.$$

We thus have $A^* \leq_{C^*} S'_1 S_1 S_0 A$ and as $S_1 S_0 A$ is well-formed we can apply the induction hypothesis once more giving

$$\mathcal{W}(S_1 S_0 A, e_2) = (S_2, t_2, b_2, C_2)$$
$$S'_1 \overline{\overline{\scriptstyle NF(S_1 S_0 A,e_2)}}\ S'_2 S_2$$
$$C^*, S'_1 S_1 S_0 A \mid e_2 : t^* \,\&\, b^*_2 \preceq^{S'_2} C_2, S_2 S_1 S_0 A \mid e_2 : t_2 \,\&\, b_2.$$

To establish (14) we set

$$S = S_2 S_1 S_0,\ t = \alpha,\ b = S_2 S_1 b_0; (S_2 b_1 + b_2)$$
$$C = S_2 S_1 C_0 \cup S_2 C_1 \cup C_2 \cup \{S_2 S_1 t_0 \subseteq \texttt{bool},\ S_2 t_1 \subseteq \alpha,\ t_2 \subseteq \alpha\}$$

for $\alpha$ the fresh variable chosen by the algorithm. Setting

$$S' = S'_2[\alpha \mapsto t^*]$$

establishes (15) since for $\gamma \in NF(A, \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)$ we have $S' S \gamma = S'_2 S_2 S_1 S_0 \gamma = S'_1 S_1 S_0 \gamma = S'_0 S_0 \gamma = S'' \gamma$. By similar reasoning, the results of applying the induction hypothesis enable us to derive

$$C^*, S''\,A \mid e_0 \,:\, \texttt{bool} \,\&\, b_0^* \preceq^{S'} S_2\,S_1\,C_0, S\,A \mid e_0 \,:\, S_2\,S_1\,t_0 \,\&\, S_2\,S_1\,b_0$$
$$C^*, S''\,A \mid e_1 \,:\, t^* \,\&\, b_1^* \preceq^{S'} S_2\,C_1, S\,A \mid e_1 \,:\, S_2\,t_1 \,\&\, S_2\,b_1$$
$$C^*, S''\,A \mid e_2 \,:\, t^* \,\&\, b_2^* \preceq^{S'} C_2, S\,A \mid e_2 \,:\, t_2 \,\&\, b_2$$

and employing that $S'\,\alpha = t^*$ it is immediate to verify

$$C^*, S''\,A \mid e \,:\, t^* \,\&\, b_0^*; (b_1^* + b_2^*) \preceq^{S'} C, S\,A \mid e \,:\, t \,\&\, b$$

which establishes (16).

**The case (sub).** The proof in (8) must have the form

$$\dfrac{\dfrac{\vdots}{jdg^- = C^*, A^* \vdash_n^{at} e \,:\, t^{*-} \,\&\, b^{*-}}}{jdg = C^*, A^* \vdash_n^{at} e \,:\, t^* \,\&\, b^*} \quad \text{(sub)}$$

where $C^* \vdash t^{*-} \subseteq t^*$ and $C^* \vdash b^{*-} \subseteq b^*$. From the induction hypothesis we have $S$, $t$, $b$, $C$ and $S'$ such that (14), (15) and (16) holds for $jdg^-$; that is

$$\mathcal{W}'(A, e) = (S, t, b, C)$$
$$S'' \underset{NF'(A,e)}{=\!=\!=} S'\,S$$
$$C^*, S''\,A \mid e \,:\, t^{*-} \,\&\, b^{*-} \preceq^{S'} C, S\,A \mid e \,:\, t \,\&\, b.$$

It immediately follows using Fact 5.3 that

$$C^*, S''\,A \mid e \,:\, t^* \,\&\, b^* \preceq^{S'} C, S\,A \mid e \,:\, t \,\&\, b$$

and this establishes (14), (15) and (16) for $jdg$.

**The case (let).** The proof tree in (8) must have the form

$$\dfrac{\dfrac{\vdots}{C^*, A^* \vdash_n^{at} e_1 \,:\, ts_1^* \,\&\, b_1^*} \qquad \dfrac{\vdots}{C^*, A^*[x : ts_1^*] \vdash_n^{at} e_2 \,:\, t_2^* \,\&\, b_2^*}}{C^*, A^* \vdash_n^{at} \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \,:\, t_2^* \,\&\, b_1^*; b_2^*} \quad \text{(let)}$$

Since $A^* \leq_{C^*} S'' A$ the induction hypothesis gives $S_1$, $t_1$, $b_1$, $C_1$, $S'_1$ such that (11), (12) and (13) holds:

$$\mathcal{W}(A, e_1) = (S_1, t_1, b_1, C_1)$$
$$S'' \overline{\overline{\scriptstyle NF(A,e_1)}}\, S'_1\, S_1$$
$$C^*, S''\, A \mid e_1 \;:\; ts_1^* \,\&\, b_1^* \;\preceq^{S'_1}\; C_1, S_1\, A \mid e_1 \;:\; ts_1 \,\&\, b_1$$
$$\text{where } ts_1 = \mathit{GEN}(S_1\, A, b_1)(C_1, t_1).$$

In particular it follows that $ts_1^* \leq_{C^*} S'_1\, ts_1$; combined with $A^* \leq_{C^*} S''\, A$ this gives $A^*[x : ts_1^*] \leq_{C^*} S'_1\, ((S_1\, A)[x : ts_1])$. As $(S_1\, A)[x : ts_1]$ is well-formed (by Lemma 4.29) we can apply the induction hypothesis to find $S_2$, $t_2$, $b_2$, $C_2$, $S'_2$ such that (11), (12) and (16) holds:

$$\mathcal{W}((S_1\, A)[x : ts_1], e_2) = (S_2, t_2, b_2, C_2)$$
$$S'_1 \overline{\overline{\scriptstyle NF((S_1\, A)[x\,:\,ts_1],\, e_2)}}\, S'_2\, S_2$$
$$C^*, S'_1\, ((S_1\, A)[x : ts_1]) \mid e_2 \;:\; t_2^* \,\&\, b_2^* \;\preceq^{S'_2}$$
$$C_2, (S_2\, S_1\, A)[x : S_2\, ts_1] \mid e_2 \;:\; t_2 \,\&\, b_2.$$

To establish (14) as well as (15) we set

$$S = S_2\, S_1,\;\; t = t_2,\;\; b = S_2\, b_1; b_2,\;\; C = S_2\, C_1 \cup C_2,\;\; S' = S'_2.$$

Our final task is to establish (16), i.e. to ensure that

$$C^*, S''\, A \mid e \;:\; t_2^* \,\&\, b_1^*; b_2^* \;\preceq^{S'_2}\; S_2\, C_1 \cup C_2, S\, A \mid e \;:\; t_2 \,\&\, S_2\, b_1; b_2$$

but this follows from the previous results, employing that $S'_2\, S_2$ equals $S'_1$ on $FV(b_1, C_1)$.

**Lifting from $\mathcal{W}'$ to $\mathcal{W}$.** As promised in the initial part of the proof we will now show the following result: let $A$ be well-formed and let $C^*$ be atomic and let $\sigma^*$ be a type; if there exists $S$, $t$, $b$, $C$, $S'$ which satisfies (14), (15) and (16), then there also exists (another) $S$, $t$, $b$, $C$, $S'$ satisfying (11), (12) and (16).

So we assume that we have $S_1, t_1, b_1, C_1$ and $S'_1$ such that

$$\mathcal{W}'(A,e) = (S_1, t_1, b_1, C_1)$$
$$S'' \underset{NF'(A,e)}{=\!=} S_1'\, S_1$$
$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \;\preceq^{S_1'}\; C_1, S_1\, A \mid e \,:\, t_1 \,\&\, b_1$$

(where $NF'(A,e)$ is the complement of the set of fresh variables generated during the call $\mathcal{W}'(A,e)$); by Lemma 4.29 $C_1$ is well-formed.

Concerning $\mathcal{F}$ it follows from Lemma 5.13 (with $R = S_1'$) that there exists $C_2$, $S_2$, and $S_2'$ such that

$$\mathcal{F}(C_1) = (S_2, C_2)$$
$$S_1' \underset{NF(C_1)}{=\!=} S_2'\, S_2$$
$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \;\preceq^{S_2'}\; C_2, S_2\, S_1\, A \mid e \,:\, S_2\, t_1 \,\&\, S_2\, b_1$$

where $NF(C_1)$ is the complement of the set of freshly generated variables in the call $\mathcal{F}(C_1)$. By Lemma 4.10, $C_2$ is atomic.

Concerning $\mathcal{R}$ it follows from Lemma 5.17 (with $R = S_2'$) that there exists $C_3$, $t_3$, and $b_3$ such that

$$\mathcal{R}(C_2, S_2\, t_1, S_2\, b_1, S_2\, S_1\, A) = (C_3, t_3, b_3)$$
$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \;\preceq^{S_2'}\; C_3, S_2\, S_1\, A \mid e \,:\, t_3 \,\&\, b_3.$$

So by letting $S = S_2\, S_1$, $t = t_3$, $b = b_3$, $C = C_3$, and $S' = S_2'$, we have

$$\mathcal{W}(A,e) = (S, t, b, C)$$
$$S'' \underset{NF(A,e)}{=\!=} S_2'\, S_2\, S_1 = S'\, S$$
$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \;\preceq^{S'}\; C, S\, A \mid e \,:\, t \,\&\, b$$

thus establishing (11), (12) and (16). Finally note (once more) that Lemma 5.8 allows us to deduce (13) from (16).

**The case (gen)**. The proof tree in (8) must have the form

$$\frac{\frac{\vdots}{C^* \cup C_0^*, A^* \vdash_n^{at} e : t^* \,\&\, b^*}}{C^*, A^* \vdash_n^{at} e : \sigma^* \,\&\, b^*} \quad \text{(gen)}$$

where

$\sigma^* = \forall(G^* : C_0^*).\ t^*$

$\forall(G^* : C_0^*).\ t^*$ is well-formed and solvable from $C^*$

so let $Dom(S_0) \subseteq G^*$ such that $C^* \vdash S_0\, C_0^*$

$G^* \cap FV(C^*, A^*, b^*) = \emptyset$.

**Phase 1:** Let $R$ be a renaming of $G^*$ into fresh variables (in particular ones which are not used by the algorithm), the need for $R$ arises since $G^*$ and $FV(S''\,A)$ are not necessarily disjoint. Let $R^{-1}$ be such that $Dom(R^{-1}) = Ran(R)$ and such that $R^{-1}\,R\,\gamma = \gamma$ for $\gamma \notin R\,G^*$. From (10) we have $A^* \leq_{C^*} S''\,A$ and as $R\,A^* = A^*$ and $R\,C^* = C^*$ we can apply Lemma 5.4 to deduce

$A^* \leq_{C^* \cup C_0^*} R\,S''\,A$.

Moreover, (9) and (8) tell us that

$C^* \cup C_0^*$ is atomic

and therefore we can apply the induction hypothesis to find $S, t, b, C$ and $S'$ such that (11), (12) and (13) hold:

$\mathcal{W}(A, e) = (S, t, b, C)$

$R\,S'' \;\overline{\overline{\scriptstyle NF(A,e)}}\; S'\,S$

$C^* \cup C_0^*, R\,S''\,A \mid e : t^* \,\&\, b^* \preceq^{S'} C, S\,A \mid e : ts \,\&\, b$

where $ts = GEN(S\,A, b)(C, t)$.

Our goal (to be accomplished in Phase 2 and 3) will be to show that

$$t^* <_{C^* \cup C_0^*} S'\, ts \tag{24}$$

implies

$$\sigma^* = \forall (G^* : C_0^*).\ t^* \leq_{C^* \cup C_0^*} S'\, ts \tag{25}$$

because then by Fact 5.3 we have

$$C^* \cup C_0^*, R\, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \preceq^{S'} C, S\, A \mid e \,:\, ts \,\&\, b$$

so by using Lemma 5.7, with the substitution $R^{-1}\, S_0$ and using $C^* \vdash S_0\, C_0^*$, we get[1]

$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \preceq^{R^{-1}\, S_0\, S'} C, S\, A \mid e \,:\, ts \,\&\, b.$$

We define $S^\dagger$ by

$$S^\dagger\, \gamma = \begin{cases} \gamma & \text{if } \gamma \in R\, G^* \\ R^{-1}\, S_0\, S'\, \gamma & \text{otherwise} \end{cases}$$

and as the variables of $R\, G^*$ are not used by the algorithm we thus have

$$C^*, S''\, A \mid e \,:\, \sigma^* \,\&\, b^* \preceq^{S^\dagger} C, S\, A \mid e \,:\, ts \,\&\, b$$

showing that $S^\dagger$ can be used to establish (13) but we must also show that $S^\dagger$ will establish (12), i.e. that

$$S''\, \gamma = S^\dagger\, S\, \gamma \text{ for } \gamma \in NF(A, e).$$

But if $\gamma$ belongs to $R\, G^*$ we have $S''\, \gamma = \gamma = S^\dagger\, \gamma = S^\dagger\, S\, \gamma$; and otherwise we have $S''\, \gamma = R^{-1}\, R\, S''\, \gamma = R^{-1}\, S_0\, R\, S''\, \gamma = R^{-1}\, S_0\, S'\, S\, \gamma = S^\dagger\, S\, \gamma$.

[1] Here we see the need for $R$, in the case $G^* \cap FV(S''\, A) \neq \emptyset$.

**Phase 2:** Returning to our proof obligation we assume (24) and must prove (25). For this we write

$$ts = \forall(G_1 : C_1).\ t$$

and let $R_1$ be a renaming of $G_1$ into fresh variables such that

$$S'\,ts = \forall(R_1\,G_1 : S'\,R_1\,C_1).\ S'\,R_1\,t.$$

Now (24) gives $S_1$ with

$$Dom(S_1) \subseteq R_1\,G_1,\ C^* \cup C_0^* \vdash S_1\,S'\,R_1\,C_1,\ C^* \cup C_0^* \vdash S_1\,S'\,R_1\,t \subseteq t^*.$$

We must show $\sigma^* \leq_{C^* \cup C_0^*} S'\,ts$, so consider $t^+$ and $C^+$ such that

$$C^+ \vdash C^* \cup C_0^* \text{ and } t^+ <_{C^+} \sigma^* = \forall(G^* : C_0^*).\ t^*$$

where the latter amounts to the existence of $S^+$ such that

$$Dom(S^+) \subseteq G^*,\ C^+ \vdash S^+\,C_0^*,\ C^+ \vdash S^+\,t^* \subseteq t^+.$$

We then have $C^+ \vdash S^+\,(C^* \cup C_0^*)$ (as $G^* \cap FV(C^*) = \emptyset$) so by Lemma 2.18 and Lemma 2.19 we get

$$C^+ \vdash S^+\,S_1\,S'\,R_1\,C_1$$
$$C^+ \vdash S^+\,S_1\,S'\,R_1\,t \subseteq S^+\,t^* \subseteq t^+.$$

Our task is to show that $t^+ <_{C^+} S'\,ts$, and for that purpose we use a trick and define $S_1^+$ by

$$S_1^+\,\gamma = \begin{cases} S^+\,S_1\,\gamma & \text{if } \gamma \in R_1\,G_1 \\ \gamma & \text{otherwise} \end{cases}$$

and our goal (to be accomplished in Phase 3) will be to show[2]

[2]Notice that a larger $G_1$ and a smaller $G^*$ makes it easier to show (26).

$$\gamma \in FV(t, C_1) \setminus G_1 \text{ implies } FV(S'\,\gamma) \cap G^* = \emptyset. \tag{26}$$

For then we for all $\gamma \in FV(t, C_1) \setminus G_1$ have (as $FV(S'\,\gamma) \cap R_1\,G_1 = \emptyset$) that $S_1^+\,S'\,R_1\,\gamma = S_1^+\,S'\,\gamma = S'\,\gamma = S^+\,S'\,\gamma = S^+\,S_1\,S'\,\gamma = S^+\,S_1\,S'\,R_1\,\gamma$ and together with the definition of $S_1^+$ this yields

$$S_1^+\,S'\,R_1\,\gamma = S^+\,S_1\,S'\,R_1\,\gamma \text{ for } \gamma \in FV(t, C_1)$$

from which we arrive at

$$C^+ \vdash S_1^+\,S'\,R_1\,C_1$$
$$C^+ \vdash S_1^+\,S'\,R_1\,t \subseteq t^+$$

which shows the desired relation $t^+ <_{C^+} S'\,ts$.

**Phase 3:** Returning to (26) we consider $\gamma \in FV(t, C_1) \setminus G_1$ and $\gamma' \in FV(S'\,\gamma)$; we must show $\gamma' \notin G^*$. Recall that

$$G_1 = Clos(FV(t), C) \setminus FV(S\,A, b)^{C\downarrow} \text{ and } C_1 = C\mid_{G_1}.$$

As $FV(t, C_1) \subseteq Clos(FV(t), C)$ it must be the case that $\gamma \in FV(S\,A, b)^{C\downarrow}$, that is there exists $\gamma_1 \in FV(S\,A, b)$ such that $C \vdash \gamma \leftarrow^* \gamma_1$. Corollary 2.31 tells us that there exists $\gamma_1' \in FV(S'\,\gamma_1)$ such that $S'\,C \vdash \gamma' \leftarrow^* \gamma_1'$.

In Phase 1 we saw that $C^* \cup C_0^*$ is atomic and hence well-formed and consistent (by Fact 4.2), and that

$$C^* \cup C_0^* \vdash S'\,C \text{ and } C^* \cup C_0^* \vdash S'\,b \subseteq b^*.$$

Therefore we by (repeated applications of) Lemma 2.32 deduce that

$$C^* \cup C_0^* \vdash \gamma' \leftarrow^* \gamma_1'$$

and moreover there exists $\gamma_2' \in FV(R\,S''\,A, b^*)$ such that

$$C^* \cup C_0^* \vdash \gamma_1' \leftarrow^* \gamma_2';$$

for if $\gamma'_1 \in FV(S'\,b)$ this follows from Lemma 2.29; and if $\gamma'_1 \in FV(S'\,S\,A)$ the result follows (with $\gamma'_2 = \gamma'_1$) since we in Phase 1 saw that $R\,S'' \underset{NF(A,e)}{=\!=} S'\,S$.

Lemma 2.33 (which can applied since $G^* \cap FV(C^*) = \emptyset$) tells us that

$$G^{*(C^* \cup C_0^*)\uparrow} = G^*$$

and[3] as $G^* \cap FV(R\,S''\,A, b^*) = \emptyset$ we infer that

$$\gamma'_2 \notin G^{*(C^* \cup C_0^*)\uparrow}$$

from which we deduce that neither $\gamma'_1$ nor $\gamma'$ belongs to $G^{*(C^* \cup C_0^*)\uparrow}$ and in particular that $\gamma' \notin G^*$. This concludes the proof of (26). □

[3] Also here we see the need for $R$, in the case $G^* \cap FV(S''\,A) \neq \emptyset$.

# Appendix E

# Proofs of Results Concerning Post-processing

**Fact 6.6** The relations $\sim$ and $\dot{\sim}$ are reflexive, symmetric, and transitive.

**Proof** Reflexiveness amounts to $I \subseteq \sim \cup \dot{\sim}$ with $I$ the identity relation; by Observation 6.5 this can be shown by establishing $I \subseteq \mathcal{G}(I \cup \sim \cup \dot{\sim})$ but this is straightforward.

Symmetry amounts to $\sim^{-1} \subseteq \sim$ and $\dot{\sim}^{-1} \subseteq \dot{\sim}$; by Observation 6.5 this can be shown by establishing

$$\sim^{-1} \subseteq \mathcal{G}(\sim \cup \sim^{-1} \cup \dot{\sim} \cup \dot{\sim}^{-1}) \text{ and}$$
$$\dot{\sim}^{-1} \subseteq \mathcal{G}(\sim \cup \sim^{-1} \cup \dot{\sim} \cup \dot{\sim}^{-1})$$

where the latter is trivial as $\dot{\sim}^{-1} \subseteq \mathcal{G}(\dot{\sim})$ can be read from Fig. 6.2.

For the former, suppose $(b_1, C_1) \sim^{-1} (b_2, C_2)$ holds; as $(\sim \cup \dot{\sim})$ is a fixed point of $\mathcal{G}$ we then have

$$(b_2, C_2)\ \mathcal{G}(\sim, \dot{\sim})\ (b_1, C_1). \qquad (1)$$

First assume $C_1 \vdash b_1 \to^{a_1} b_1'$, due to (1) there exists $a_2$ and $b_2'$ such that $C_2 \vdash b_2 \to^{a_2} b_2'$ with $(a_1, C_1) \dot{\sim}^{-1} (a_2, C_2)$ and $(b_1', C_1) \sim^{-1} (b_2', C_2)$.

Next assume $C_2 \vdash b_2 \rightarrow^{a_2} b_2'$, due to (1) there exists $a_1$ and $b_1'$ such that $C_1 \vdash b_1 \rightarrow^{a_1} b_1'$ with $(a_1, C_1) \dot{\sim}^{-1} (a_2, C_2)$ and $(b_1', C_1) \sim^{-1} (b_2', C_2)$.

This demonstrates $\sim^{-1} \subseteq \mathcal{G}(\sim^{-1} \cup \dot{\sim}^{-1})$, as desired.

Transitivity amounts to $(\sim \circ \sim) \subseteq \sim$ and $(\dot{\sim} \circ \dot{\sim}) \subseteq \dot{\sim}$; by Observation 6.5 this can be shown by establishing

$$\sim \circ \sim \subseteq \mathcal{G}(\sim \circ \sim \cup \dot{\sim} \circ \dot{\sim}) \text{ and}$$
$$\dot{\sim} \circ \dot{\sim} \subseteq \mathcal{G}(\dot{\sim})$$

where the latter can be trivially read from Fig. 6.2.

For the former, suppose $(b_1, C_1) \sim \circ \sim (b_2, C_2)$ holds because $(b_1, C_1) \sim (b_3, C_3)$ and $(b_3, C_3) \sim (b_2, C_2)$; as $(\sim \cup \dot{\sim})$ is a fixed point of $\mathcal{G}$ we then have

$$(b_1, C_1)\ \mathcal{G}(\sim, \dot{\sim})\ (b_3, C_3) \text{ and} \tag{2}$$
$$(b_3, C_3)\ \mathcal{G}(\sim, \dot{\sim})\ (b_2, C_2). \tag{3}$$

First assume $C_1 \vdash b_1 \rightarrow^{a_1} b_1'$, due to (2) there exists $a_3$ and $b_3'$ such that $C_3 \vdash b_3 \rightarrow^{a_3} b_3'$ with $(a_1, C_1) \dot{\sim} (a_3, C_3)$ and $(b_1', C_1) \sim (b_3', C_3)$, and due to (3) there then exists $a_2$ and $b_2'$ such that $C_2 \vdash b_2 \rightarrow^{a_2} b_2'$ with $(a_3, C_3) \dot{\sim} (a_2, C_2)$ and $(b_3', C_3) \sim (b_2', C_2)$, that is $(a_1, C_1) \dot{\sim} \circ \dot{\sim} (a_2, C_2)$ and $(b_1', C_1) \sim \circ \sim (b_2', C_2)$.

Next assume $C_2 \vdash b_2 \rightarrow^{a_2} b_2'$, due to (3) there exists $a_3$ and $b_3'$ such that $C_3 \vdash b_3 \rightarrow^{a_3} b_3'$ with $(a_3, C_3) \dot{\sim} (a_2, C_2)$ and $(b_3', C_3) \sim (b_2', C_2)$, and due to (2) there then exists $a_1$ and $b_1'$ such that $C_1 \vdash b_1 \rightarrow^{a_1} b_1'$ with $(a_1, C_1) \dot{\sim} (a_3, C_3)$ and $(b_1', C_1) \sim (b_3', C_3)$, that is $(a_1, C_1) \dot{\sim} \circ \dot{\sim} (a_2, C_2)$ and $(b_1', C_1) \sim \circ \sim (b_2', C_2)$.

This demonstrates $\sim \circ \sim \subseteq \mathcal{G}(\sim \circ \sim \cup \dot{\sim} \circ \dot{\sim})$, as desired. □

**Lemma 6.9** Let $C$ be a set of behaviour constraints, and let $\mathcal{S}_F$ be a homomorphism with the following properties:

1. if for some $b_1'$ and $b_2$ it holds that $(b_1' \subseteq \mathcal{S}_F(b_2)) \in \mathcal{S}_F(C)$, then there exists $b_1$ with $\mathcal{S}_F(b_1) = b_1'$ such that $C \vdash b_1 \subseteq b_2$;
2. if for some $\beta$ it holds that $F(\beta)$ is not a variable, then

$$C \vdash F(\beta) \subseteq \beta \text{ and } \mathcal{S}_F(F(\beta)) = F(\beta).$$

We then have the following implications:

1. if $\mathcal{S}_F(C) \vdash b'_1 \subseteq \mathcal{S}_F(b_2)$ there exists $b_1$ with $\mathcal{S}_F(b_1) = b'_1$ such that $C \vdash b_1 \subseteq b_2$;

2. if $\mathcal{S}_F(C) \vdash \mathcal{S}_F(b_2) \to^{a'} b'_0$ there exists $a$, $b_0$ with $\mathcal{S}_F(a) = a'$ and $\mathcal{S}_F(b_0) = b'_0$ such that $C \vdash b_2 \to^{a} b_0$.

**Proof** We first prove 1.

As $\mathcal{S}_F(C)$ is consistent (Fact 5.11), Corollary 2.28 tells us that $\mathcal{S}_F(C) \vdash_{fw} b'_1 \subseteq \mathcal{S}_F(b_2)$; we shall perform induction in this derivation (in the various cases, $b_1$ and $b'_1$ and $b_2$ always retain their meaning from the lemma formulation).

**The case (axiom).** Follows from Property 1.

**The case (refl).** As $b_1$ we can choose $b_2$.

**The case (trans).** Suppose $\mathcal{S}_F(C) \vdash b'_1 \subseteq \mathcal{S}_F(b_2)$ because $\mathcal{S}_F(C) \vdash b'_1 \subseteq b'_3$ and $\mathcal{S}_F(C) \vdash b'_3 \subseteq \mathcal{S}_F(b_2)$. By applying the induction hypothesis on the latter inference we find $b_3$ with $\mathcal{S}_F(b_3) = b'_3$ such that $C \vdash b_3 \subseteq b_2$; by applying the induction hypothesis on the former inference we next find $b_1$ with $\mathcal{S}_F(b_1) = b'_1$ such that $C \vdash b_1 \subseteq b_3$; as then $C \vdash b_1 \subseteq b_2$ this yields the claim.

**The case (cong).** Suppose $\mathcal{S}_F(C) \vdash b'_1 = b'_{11}; b'_{12} \subseteq b'_{21}; b'_{22} = \mathcal{S}_F(b_2)$ because

$$\mathcal{S}_F(C) \vdash b'_{11} \subseteq b'_{21} \text{ and } \mathcal{S}_F(C) \vdash b'_{12} \subseteq b'_{22}. \tag{4}$$

First assume that $b_2$ takes the form $b_{21}; b_{22}$, then $\mathcal{S}_F(b_{21}) = b'_{21}$ and $\mathcal{S}_F(b_{22}) = b'_{22}$; we can thus apply the induction hypothesis twice to find $b_{11}$ and $b_{12}$ with $\mathcal{S}_F(b_{11}) = b'_{11}$ and $\mathcal{S}_F(b_{12}) = b'_{12}$ such that $C \vdash b_{11} \subseteq b_{21}$ and $C \vdash b_{12} \subseteq b_{22}$. Now define $b_1 = b_{11}; b_{12}$ and it is easy to verify that $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \subseteq b_2$.

Next assume that $b_2$ is not of the form $\_;\_$, then it must be the case that $b_2$ is a variable and that $F(b_2) = b'_{21}; b'_{22}$. As Property 2 holds we have $C \vdash F(b_2) \subseteq b_2$ and $\mathcal{S}_F(b'_{21}; b'_{22}) = b'_{21}; b'_{22}$, implying $\mathcal{S}_F(b'_{21}) = b'_{21}$ and $\mathcal{S}_F(b'_{22}) = b'_{22}$. We can thus apply the induction hypothesis twice on (4) to find $b_{11}$ and $b_{12}$ with $\mathcal{S}_F(b_{11}) = b'_{11}$ and $\mathcal{S}_F(b_{12}) = b'_{12}$ such that $C \vdash b_{11} \subseteq b'_{21}$ and $C \vdash b_{12} \subseteq b'_{22}$. Now define $b_1 = b_{11}; b_{12}$ and it is easy to verify that $\mathcal{S}_F(b_1) = b'_1$; moreover we have

$$C \vdash b_1 \subseteq b'_{21}; b'_{22} = F(b_2) \subseteq b_2$$

as desired.

The rules for $+$ and $SPAWN$ are treated in a similar way.

**The case (seq-ass).** This is really two rules (as $b \equiv b'$ amounts to $b \subseteq b'$ and $b' \subseteq b$), we shall consider only one of them as the other is similar. So suppose $b'_1 = b'_3; (b'_4; b'_5)$ and $\mathcal{S}_F(b_2) = (b'_3; b'_4); b'_5$.

First assume that there exists $b_3$, $b_4$ and $b_5$ such that $b_2 = (b_3; b_4); b_5$, then $\mathcal{S}_F(b_3) = b'_3$ and $\mathcal{S}_F(b_4) = b'_4$ and $\mathcal{S}_F(b_5) = b'_5$; this shows that we can use $b_1 = b_3; (b_4; b_5)$.

Next assume that there exists $b_6$ and $b_5$ such that $b_2 = b_6; b_5$ but $b_6$ is not of the form $\_;\_$, then we infer that $F(b_6) = b'_3; b'_4$ with $b_6$ a variable and that $\mathcal{S}_F(b_5) = b'_5$. As Property 2 holds we have $C \vdash b'_3; b'_4 \subseteq b_6$ and $\mathcal{S}_F(F(b_6)) = F(b_6)$, implying $\mathcal{S}_F(b'_3) = b'_3$ and $\mathcal{S}_F(b'_4) = b'_4$. By defining $b_1$ as $b'_3; (b'_4; b_5)$ we obtain the desired relations: $\mathcal{S}_F(b_1) = b'_1$, and

$$C \vdash b_1 \equiv (b'_3; b'_4); b_5 \subseteq b_6; b_5 = b_2.$$

Finally assume that $b_2$ is not of the form $\_;\_$, then we infer that $F(b_2) = (b'_3; b'_4); b'_5$ with $b_2$ a variable. As Property 2 holds we have $C \vdash F(b_2) \subseteq b_2$, together with $\mathcal{S}_F(b'_3) = b'_3$ and $\mathcal{S}_F(b'_4) = b'_4$ and $\mathcal{S}_F(b'_5) = b'_5$. By defining $b_1$ as $b'_3; (b'_4; b'_5)$ we obtain the desired relations: $\mathcal{S}_F(b_1) = b'_1$, and $C \vdash b_1 \equiv F(b_2) \subseteq b_2$.

**The case (seq-neut).** It is enough to consider the rules for $\varepsilon; \_$, as the rules for $\_; \varepsilon$ can be treated in an analogous way. For one direction ($\varepsilon; b \subseteq b$), suppose that $b'_1 = \varepsilon; \mathcal{S}_F(b_2)$; then we can use $b_1 = \varepsilon; b_2$ as $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \equiv b_2$.

For the other direction ($b \subseteq \varepsilon; b$), suppose that $\mathcal{S}_F(b_2) = \varepsilon; b'_1$.

First assume that there exists $b_1$ such that $b_2 = \varepsilon; b_1$. As desired we then have $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \equiv b_2$.

Next assume that there exists $b_0$ and $b_1$ such that $b_2 = b_0; b_1$ but $b_0 \neq \varepsilon$, then $F(b_0) = \varepsilon$ with $b_0$ a variable and $\mathcal{S}_F(b_1) = b'_1$. As Property 2 holds we have $C \vdash \varepsilon \subseteq b_0$, enabling us to show the desired

$$C \vdash b_1 \equiv \varepsilon; b_1 \subseteq b_0; b_1 = b_2.$$

Finally assume that $b_2$ is not of the form _; _ and hence a variable; as Property 2 holds we have $C \vdash \varepsilon; b'_1 \subseteq b_2$ and $\mathcal{S}_F(b'_1) = b'_1$. By defining $b_1 = b'_1$ we therefore obtain the desired relations: $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \equiv \varepsilon; b'_1 \subseteq b_2$.

**The case (ub).** Suppose $\mathcal{S}_F(b_2) = b'_1 + \_$ (the case where $\mathcal{S}_F(b_2) = \_ + b'_1$ is similar).

First assume that there exists $b_1$ such that $b_2$ takes the form $b_1 + \_$. As desired we then have $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \subseteq b_2$.

Next assume that $b_2$ is not of the form _ + _ and hence a variable; as Property 2 holds we have $C \vdash \mathcal{S}_F(b_2) \subseteq b_2$ and $\mathcal{S}_F(b'_1) = b'_1$. By defining $b_1 = b'_1$ we therefore obtain the desired relations: $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \subseteq \mathcal{S}_F(b_2) \subseteq b_2$.

**The case (lub).** Suppose $\mathcal{S}_F(C) \vdash b'_1 = b'_{11} + b'_{12} \subseteq \mathcal{S}_F(b_2)$ because $\mathcal{S}_F(C) \vdash b'_{11} \subseteq \mathcal{S}_F(b_2)$ and $\mathcal{S}_F(C) \vdash b'_{12} \subseteq \mathcal{S}_F(b_2)$. We can apply the induction hypothesis twice to find $b_{11}$ and $b_{12}$ with $\mathcal{S}_F(b_{11}) = b'_{11}$ and $\mathcal{S}_F(b_{12}) = b'_{12}$ such that $C \vdash b_{11} \subseteq b_2$ and $C \vdash b_{12} \subseteq b_2$. Now define $b_1 = b_{11} + b_{12}$ and it is easy to verify that $\mathcal{S}_F(b_1) = b'_1$ and $C \vdash b_1 \subseteq b_2$.

**This concludes the proof of 1. We next prove 2:** suppose $\mathcal{S}_F(C) \vdash \mathcal{S}_F(b_2) \to^{a'} b'_0$, that is $\mathcal{S}_F(C) \vdash a'; b'_0 \subseteq \mathcal{S}_F(b_2)$, then by 1 there exists $b_1$ with $\mathcal{S}_F(b_1) = a'; b'_0$ such that $C \vdash b_1 \subseteq b_2$.

First assume that $b_1$ does not take the form _; _ and thus is a variable; as Property 2 holds we have $C \vdash a'; b'_0 \subseteq b_1$ and $\mathcal{S}_F(a') = a'$ and $\mathcal{S}_F(b'_0) = b'_0$. Define $a = a'$ and $b_0 = b'_0$, then we have the desired relations $\mathcal{S}_F(a) = a'$ and $\mathcal{S}_F(b_0) = b'_0$ and $C \vdash a; b_0 \subseteq b_1 \subseteq b_2$, that is $C \vdash b_2 \to^a b_0$.

Next assume that $b_1$ takes the form $a; b_0$. As desired we then have $\mathcal{S}_F(a) = a'$ and $\mathcal{S}_F(b_0) = b'_0$ and $C \vdash a; b_0 \subseteq b_2$.

Finally assume that $b_1$ takes the form $b; b_0$ with $b$ not an action, as $\mathcal{S}_F(b) = a'$ we deduce that $b$ must be a variable. As Property 2 holds we have $C \vdash a' \subseteq b$ and $\mathcal{S}_F(a') = a'$, so by letting $a = a'$ we obtain the desired relations $\mathcal{S}_F(a) = a'$ and $\mathcal{S}_F(b_0) = b'_0$ and $C \vdash a; b_0 \subseteq b; b_0 \subseteq b_2$, that is $C \vdash b_2 \to^a b_0$. □

# Appendix F

# List of Symbols

| letter(s) | denotes |
|---|---|
| $A$ | environment |
| $a$ | action |
| $\alpha$ | type variable |
| $b$ | behaviour |
| $\beta$ | behaviour variable |
| $C$ | constraint set |
| $ca$ | channel action |
| $ch$ | channel identifier |
| $E$ | evaluation context |
| $e$ | expression in *Exp* or *EExp* |
| $g$ | type/behaviour/region |
| $\gamma$ | type/behaviour/region variable |
| $jdg$ | typing judgement |
| $p$ | process identifier |
| $PB$ | mapping from process identifiers to behaviours |
| $PP$ | process pool |
| $PT$ | mapping from process identifiers to types |
| $R$ | ML substitution or special kind of substitution, e.g. matching substitution or "renaming" substitution |
| $r$ | region |
| $\rho$ | region variable |
| $S$ | substitution |
| $\sigma$ | type or type scheme |
| $sa$ | semantic action |
| $sh$ | shape (of type) |
| $t$ | type |
| $ts$ | type scheme |
| $u$ | ML type |
| $us$ | ML type scheme |
| $w$ | weakly evaluated expression |

Table F.1: Naming conventions.

| judgement | explanation |
|---|---|
| $C \vdash t_1 \subseteq t_2$ | Figure 2.6 |
| $C \vdash b_1 \subseteq b_2$ | Figure 2.7 |
| $C \vdash r_1 \subseteq r_2$ | Figure 2.8 |
| $C \vdash_{dc} g_1 \subseteq g_2$ | Definition 2.7 |
| $C \vdash_{fw} g_1 \subseteq g_2$ | $C \vdash g_1 \subseteq g_2$ via forward derivation |
| $C \vdash g_1 \equiv g_2$ | $C \vdash g_1 \subseteq g_2$ and $C \vdash g_2 \subseteq g_1$ |
| $C, A \vdash e : \sigma \,\&\, b$ | Figure 2.5 |
| $C, A \vdash_n e : \sigma \,\&\, b$ | Definition 2.22 |
| $C^*, A^* \vdash^{at} e : \sigma^* \,\&\, b^*$ | atomic inference |
| $C, A \mid e : \sigma \,\&\, b$ | typing judgement |
| $C \vdash \gamma \leftarrow \beta$ | Definition 2.8 |
| $A' \vdash^{\mathrm{ML}} e : u$ | Figure 2.9 |
| $A' \vdash^{\mathrm{ML}}_n e : u$ | As Definition 2.22 |

Table F.2: Judgements.

| transition | explanation |
|---|---|
| $e \to e'$ | Definition 3.5 |
| $e \rightharpoonup e'$ | Definition 3.5 |
| $PP \overset{sa}{\Leftrightarrow\!\!\rightarrow} PP'$ | Definition 3.11 |
| $C \rightharpoonup C'$ | Figure 4.2 |
| $(S, C, \sim) \Leftrightarrow\!\!\rightarrow (S', C', \sim')$ | Figure 4.3 |
| $A \vdash (C, t, b) \Leftrightarrow\!\!\rightarrow (C', t', b')$ | Figure 4.5 |
| $C \vdash b \to^{a} b'$ | Definition 6.4 |

Table F.3: Transitions and rewritings.

| relation | explanation |
| --- | --- |
| $t_1 \approx t_2$ | Definition 5.9 |
| $t <_C ts$ | Definition 5.1 |
| $\sigma_1 \leq_C \sigma_2$ | Definition 5.2 |
| $jdg_1 \preceq^S jdg_2$ | Definition 5.5 |
| $S_1 \underset{X}{=} S_2$ | $\forall \gamma \in X : S_1\,\gamma = S_2\,\gamma$ |
| $(b, C) \sim (b', C')$ | Figure 6.1 |
| $(a, C) \dot{\sim} (a', C')$ | Figure 6.2 |
| $u \prec^R_\epsilon ts$ | Definition A.7 |
| $u \prec us$ | Definition A.8 |
| $us \cong^R_\epsilon ts$ | Definition A.9 |
| $u \cong^R_\epsilon t$ | Definition A.11 |
| $\gamma \sim_C \gamma'$ | used in defining $Clos(,)$ |

Table F.4: Relations.

| operation | explanation |
|---|---|
| $C^b$ | behaviour constraints in $C$ |
| $C^r$ | region constraints in $C$ |
| $C^t$ | type constraints in $C$ |
| $\overline{C}$ | Definition 2.7 |
| $X^{C\downarrow}$ | Definition 2.9 |
| $X^{C\uparrow}$ | Definition 2.9 |
| $C \dot{\cup} C'$ | $C \cup C'$ in case $C \cap C' = \emptyset$ |
| $C \mid_{\{\vec{\alpha}\vec{\beta}\vec{\rho}\}}$ | $\{(g_1 \subseteq g_2) \in C \mid FV(g_1, g_2) \cap \{\vec{\alpha}\vec{\beta}\vec{\rho}\} \neq \emptyset\}$ |

Table F.5: Operations on constraints.

| symbol | explanation |
|---|---|
| $R \models C$ | $\forall (r_1 \subseteq r_2) \in C : R(r_1) \subseteq R(r_2)$ |
| $ca^r$ | the region part of the channel action $ca$ |
| $(\gamma \Leftarrow^* \gamma') \in C$ | Definition 4.12 |
| $E[e]$ | filling $e$ into the hole in the evaluation context $E$ |
| $sh[\vec{t}, \vec{\beta}, \vec{\rho}]$ | filling $\vec{t}$,$\vec{\beta}$,$\vec{\rho}$ into the holes in the shape $sh$. |

Table F.6: Miscellaneous.

# Index

# Bibliography

[1] Torben Amtoft and Flemming Nielson and Hanne Riis Nielson: Type and behaviour reconstruction for higher-order concurrent programs. *Journal of Functional Programming*, 7(3):321–347, May 1997.

[2] Torben Amtoft and Flemming Nielson and Hanne Riis Nielson and Jürgen Ammann: Polymorphic subtypes for effect analysis: the dynamic semantics. In *Analysis and Verification of Multiple-Agent Languages*, pages 172–206, SLNCS 1192, 1997.

[3] Torben Amtoft and Hanne Riis Nielson and Flemming Nielson: Behaviour analysis for validating communication patterns. Technical Report DAIMI PB-527, Dept. of Comp. Science, Aarhus University, September 1997. Invited submission for Springer International Journal on *Software Tools for Technology Transfer.*

[4] Luis Damas and Robin Milner: Principal type-schemes for functional programs. In *Proc. of POPL '82*, ACM Press, 1982.

[5] You-Chin Fuh and Prateek Mishra: Polymorphic subtype inference: Closing the theory-practice gap. In *Proc. TAPSOFT '89*, pages 167–183, SLNCS 352, 1989.

[6] You-Chin Fuh and Prateek Mishra: Type inference with subtypes. *Theoretical Computer Science* 73, pages 155–175, 1990.

[7] Fritz Henglein and Christian Mossin: Polymorphic binding-time analysis. In *Proc. ESOP '94*, pages 287–301, SLNCS 788, 1994.

[8] Mark P. Jones: A theory of qualified types. In *Proc. ESOP '92*, pages 287–306, SLNCS 582, 1992.

[9] Pierre Jouvelot and David K. Gifford: Algebraic reconstruction of types and effects. In *Proc. POPL'91*, pages 303–310. ACM Press, 1991.

[10] Xavier Leroy and Pierre Weis: Polymorphic type inference and assignment. In *Proc. POPL '91*, pages 291–302. ACM Press, 1991.

[11] Robin Milner: A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[12] John C. Mitchell: Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, 1991.

[13] Flemming Nielson (editor): ML with Concurrency: Design, Analysis, Implementation and Application. Springer Monographs in Computer Science, 1997.

[14] Flemming Nielson and Hanne Riis Nielson: Constraints for polymorphic behaviours for Concurrent ML. In *Proc. CCL'94*, SLNCS 845, 1994.

[15] Hanne Riis Nielson and Torben Amtoft and Flemming Nielson: Behaviour analysis and safety conditions: a case study in CML. Accepted for presentation at *Nordic Workshop on Programming Theory* (NWPT'97), Tallinn, Estonia, October 1997.

[16] Hanne Riis Nielson and Flemming Nielson: Automatic binding analysis for a typed $\lambda$-calculus. *Science of Computer Programming*, 10:139–176, 1988.

[17] Hanne Riis Nielson and Flemming Nielson: Higher-order concurrent programs with finite communication topology. In *Proc. POPL'94*, pages 84–97. ACM Press, 1994. Full version appears as [18].

[18] Hanne Riis Nielson and Flemming Nielson. Communication analysis for Concurrent ML. In [13].

[19] Hanne Riis Nielson and Flemming Nielson and Torben Amtoft: Polymorphic subtypes for effect analysis: the static semantics. In *Analysis and Verification of Multiple-Agent Languages*, pages 141–171, SLNCS 1192, 1997.

[20] Prakash Panangaden and John H. Reppy: The essence of Concurrent ML. In [13].

[21] Gordon D. Plotkin: A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, Denmark, 1981.

[22] John H. Reppy: Concurrent ML: Design, application and semantics. In *Proc. Functional Programming, Concurrency, Simulation and Automated Reasoning*, pages 165–198, SLNCS 693, 1993.

[23] J.A. Robinson: A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[24] Jörg H. Siekmann: Unification theory. *J. Symbolic Computation*, 7:207–274, 1989.

[25] Geoffrey S. Smith: Polymorphic type inference for languages with overloading and subtyping. Ph.D thesis from Cornell, 1991.

[26] Geoffrey S. Smith: Polymorphic type inference with overloading and subtyping. In *Proc. TAPSOFT '93*, pages 671–685, SLNCS 668, 1993. Also see: Principal type schemes for functional programs with overloading and subtyping: *Science of Computer Programming* 23, pages 197–226, 1994.

[27] Jean-Pierre Talpin and Pierre Jouvelot: Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–271, 1992.

[28] Jean-Pierre Talpin and Pierre Jouvelot: The type and effect discipline. *Information and Computation*, 111, 1994. (A preliminary version appeared in *Proc. LICS '92*, pages 162–173.)

[29] Yan-Mei Tang: Control flow analysis by effect systems and abstract interpretation. PhD thesis, Ecoles des Mines de Paris, 1994.

[30] Mads Tofte: Type inference for polymorphic references. *Information and Computation*, 89:1–34, 1990.

[31] Mads Tofte and Lars Birkedal: Region-annotated types and type schemes, 1996. Submitted for publication.

[32] Andrew K. Wright: Typing references by effect inference. In *Proc. ESOP '92*, pages 473–491, SLNCS 582, 1992.

[33] Andrew K. Wright and Matthias Felleisen: A syntactic approach to type soundness. *Information and Computation*, 115, pages 38–94, 1994.