Syntactic Control of Interference
Part 2

John C. Reynolds[1]

April 17, 1989

CMU-CS-89-130

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This is a preprint of a paper that will appear in the Proceedings of the 16th International Colloquium on Automata, Languages, and Programming (Stresa, July 11–15, 1989), to be published in the Springer-Verlag Lecture Notes in Computer Science.

Abstract

In 1978, we proposed that Algol-like languages should be constrained so that aliasing between variables and, more generally, interference between commands or procedures would be syntactically detectable in a fail-safe manner. In particular, we proposed syntactic restrictions that prohibited interference between distinct identifiers, while permitting interference between qualifications of the same identifier. However, these restrictions had the unfortunate property that syntactic correctness was not preserved by beta reduction.

In the present paper, we show how this difficulty can be avoided by the use of a variant of conjunctive types. We also give an algorithm for typechecking explicitly typed programs.

---

[1]Research supported by NSF Grant CCR-8620191. A portion of the research was also sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, under contract number F33615-87-C-1499, monitored by the Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, Ohio. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any agency of the US Government.

1. Introduction

Whenever a programming language combines assignment with a sufficiently powerful procedure mechanism, the phenomenon of "aliasing" appears, as well as various anomalies that are often called "interfering side effects". These are all instances of the general phenomenon of *interference*: for example, two (phrases denoting) variables interfere if assigning to either one can affect the value of the other, two commands interfere if either one assigns to a variable that is evaluated or assigned to by the other, and two procedures interfere if either one assigns to a global variable that is evaluated or assigned to by the other.

Interference is not always undesirable; procedures that manipulate common global variables are widely used in programming, and collections of such procedures are the essence of "object-oriented" programming. But it would be desirable to constrain a programming language so that interference is syntactically detectable (in a fail-safe sense). In particular, such a constraint is necessary in a language that provides concurrent processing with shared variables, in order to enforce the protection of the shared variables by critical regions [4,1,3].

Eleven years ago, in [7], I proposed syntactic constraints to make interference detectable that were based on three principles:

- If no identifier occurring free in the phrase p interferes with any identifier occurring free in the phrase q, then p does not interfere with q.

In effect, all "channels" of interference must be named by identifiers.

- Distinct identifiers do not interfere.

One can still have interfering procedures (or other entities), but they must occur within a single object or, in other words, be named by different qualifications of the same identifier.

- *Passive* phrases, which perform no assignment or other actions that could cause interference, do not interfere with one another.

Passive phrases include both (side-effect-free) expressions and procedures that do not assign to global variables.

Unfortunately, the specific syntactic constraints described in [7] have the unhappy consequence that certain legal phrases beta-reduce to illegal phrases. In the present paper, we will use conjunctive types [2] to define constraints, based on the above principles, that avoid this problem. The essential change is that, instead of focusing on a relation between

phrases (denoted by # in [7]) that asserts that the phrases do not interfere, we will focus on a relation between type assignments (denoted by $\perp$ and called *independence*) that asserts that the capabilities represented by the type assignments cannot cause interference.

2. An Illustrative Language

To make our exposition concrete, we will use an Algol-like illustrative language [5] that is an extended lambda calculus with construction and selection operations for named tuples, a conditional construct, and some of the various operations for expressions and commands that are typically found in imperative languages. We will also introduce an operator $\|$ that executes two command concurrently; our goal is to prohibit interference between the operands of $\|$, so that the semantics of our language will be determinate. (In a more realistic language, indeterminacy would be permitted, but only under the control of critical regions.)

The following productions define the untyped abstract syntax of our language:

⟨phrase⟩ ::= ⟨identifier⟩	identifiers
| λ⟨identifier⟩:⟨finite set of types⟩. ⟨phrase⟩	abstraction
| ⟨phrase⟩⟨phrase⟩	application
| $\langle$⟨identifier⟩ ≡ ⟨phrase⟩, ... , ⟨identifier⟩ ≡ ⟨phrase⟩$\rangle$	tupling
| ⟨phrase⟩.⟨identifier⟩	selection
| **if** ⟨phrase⟩ **then** ⟨phrase⟩ **else** ⟨phrase⟩	conditionals
| 0 | 0.5 | ⟨phrase⟩ + ⟨phrase⟩	expressions
| ⟨phrase⟩ := ⟨phrase⟩ | ⟨phrase⟩ ; ⟨phrase⟩ | **while** ⟨phrase⟩ **do** ⟨phrase⟩	commands
| ⟨phrase⟩ || ⟨phrase⟩	concurrency

Of course in a real language there would be additional operations for expressions and commands, but such operations are so similar to those we have included that they would add nothing but length to our exposition. On the other hand, there are language features, such as multiargument procedures, **let** definitions, recursion, and variable declarations, that we have omitted since they are syntactic sugar that can be defined in terms of the above language plus appropriate built-in procedures [5].

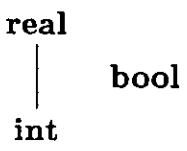
This illustrative language is similar to the recently proposed Forsythe language [6], except that it lacks the escape operator, the merging operation, and the treatment of assignment as a procedure call that occur in Forsythe.

Notice that lambda expressions contain explicit type information, which will be used to make typechecking feasible.

3. Types

As in [5] and [6], we distinguish between a *data type*, such as "integer" and "Boolean", which denotes a set of values appropriate to some kind of variable, and a *phrase type*, such as "integer expression" or "proper procedure accepting an integer expression", which denotes a set (or domain) of meanings appropriate to some kind of phrase. (The unqualified term "type" will always mean "phrase type".)

We assume that the set of data types is equipped with a preorder $\leq_{\text{data}}$, and say that δ is a *subtype* of δ' when $\delta \leq_{\text{data}} \delta'$. Specifically, we assume that **int**(eger), **real**, and **bool**(ean) are data types, and that **int** $\leq_{\text{data}}$ **real**:



More generally, we assume that the set of data types is equipped with two binary operations $\dot{\sqcup}$ and $\dot{\sqcap}$ such that $\delta_1 \dot{\sqcup} \delta_2$ ($\delta_1 \dot{\sqcap} \delta_2$) is a finite complete set of upper (lower) bounds of δ_1 and δ_2, i.e.

If $\delta \in \delta_1 \dot{\sqcup} \delta_2$ then $\delta_1 \leq_{\text{data}} \delta$ and $\delta_2 \leq_{\text{data}} \delta$.

If $\delta_1 \leq_{\text{data}} \delta$ and $\delta_2 \leq_{\text{data}} \delta$ then there is a $\delta_0 \in \delta_1 \dot{\sqcup} \delta_2$ such that $\delta_0 \leq_{\text{data}} \delta$.

If $\delta \in \delta_1 \dot{\sqcap} \delta_2$ then $\delta \leq_{\text{data}} \delta_1$ and $\delta \leq_{\text{data}} \delta_2$.

If $\delta \leq_{\text{data}} \delta_1$ and $\delta \leq_{\text{data}} \delta_2$ then there is a $\delta_0 \in \delta_1 \dot{\sqcap} \delta_2$ such that $\delta \leq_{\text{data}} \delta_0$.

For phrase types, we use the canonical formalism for conjunctive types [6], in which there is no explicit conjunction operator, but certain contexts require an identifier or phrase to have all types belonging to some finite set, rather than a single type.

Corresponding to each data type δ, there are two phrase types: δ **exp**(ression), describing phrases that can be evaluated to obtain a value of data type δ, and δ **acc**(eptor), describing phrases whose execution can accept a value of data type δ. (What is usually called a δ variable is a phrase having both of the types δ **exp** and δ **acc**.) There is one additional primitive phrase type: **comm**(and).

If an object possesses a field of type θ that is named by an identifier ι, then the object has the type $\iota{:}\theta$. Notice that no type describes more than one field; instead an object with several fields has several types, each describing a single field.

If $\hat{\theta}$ is a finite set of types and θ is a type, then $\hat{\theta} \longrightarrow \theta$ is a type describing a procedure whose call will have type θ when its parameter has all of the types in $\hat{\theta}$. Moreover, if such

a procedure causes no assignment to a global variable, it will also have the type $\hat{\theta} \xrightarrow[P]{} \theta$ and be said to be a *passive* procedure.

We will partition the set of phrase types into passive and active types. First, however, we note that, without loss of generality, we can require a procedure whose calls are passive to itself be passive and to have a passive parameter. Putting the matter the other way round, we require that in $\hat{\theta} \longrightarrow \theta$, θ must be active (i.e. not passive), and in $\hat{\theta} \xrightarrow[P]{} \theta$, θ must be active if any member of $\hat{\theta}$ is active.

Henceforth, we will use the following metavariables:

δ: data types
ϕ: passive phrase types
α: active phrase types
θ: arbitrary phrase types
$\hat{\phi}$: finite sets of passive phrase types
$\hat{\theta}$: finite sets of arbitrary phrase types
ι: identifiers .

(For brevity, we will call a finite set of phrase types *passive* when its members are all passive.) Then the sets of passive and active phrase types may be defined grammatically:

$$\phi ::= \delta\ \mathbf{exp} \mid \hat{\phi} \xrightarrow[P]{} \phi \mid \hat{\theta} \xrightarrow[P]{} \alpha \mid \iota{:}\,\phi$$
$$\alpha ::= \delta\ \mathbf{acc} \mid \mathbf{comm} \mid \hat{\theta} \longrightarrow \alpha \mid \iota{:}\,\alpha$$
$$\theta ::= \phi \mid \alpha$$

The subtype preorder is defined for phrase types and for finite sets of phrase types by mutual recursion. For phrase types,

$$\delta\ \mathbf{exp} \leq \delta'\ \mathbf{exp} \text{ when } \delta \leq_{\text{data}} \delta'$$

$$\delta\ \mathbf{acc} \leq \delta'\ \mathbf{acc} \text{ when } \delta' \leq_{\text{data}} \delta$$

$$\mathbf{comm} \leq \mathbf{comm}$$

$$\left.\begin{array}{l} \hat{\theta} \xrightarrow[P]{} \theta \leq \hat{\theta}' \xrightarrow[P]{} \theta' \\ \hat{\theta} \xrightarrow[P]{} \theta \leq \hat{\theta}' \longrightarrow \theta' \\ \hat{\theta} \longrightarrow \theta \leq \hat{\theta}' \longrightarrow \theta' \end{array}\right\} \text{ when } \hat{\theta}' \leq \hat{\theta} \text{ and } \theta \leq \theta'$$

$$\iota{:}\,\theta \leq \iota{:}\,\theta' \text{ when } \theta \leq \theta' ,$$

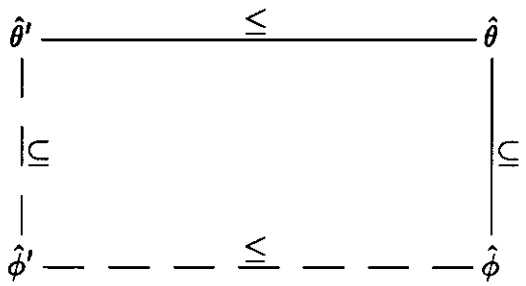
and in any other case $\theta \leq \theta'$ is false. For finite sets of phrase types,

$$\hat{\theta} \leq \hat{\theta}' \text{ when } (\forall \theta' \in \hat{\theta}')(\exists \theta \in \hat{\theta})\ \theta \leq \theta' .$$

(Note that $\hat{\theta}' \subseteq \hat{\theta}$ implies $\hat{\theta} \leq \hat{\theta}'$.) This definition of subtype implies

Proposition 1 *If ϕ is passive and $\theta \leq \phi$ then θ is passive.*

Proposition 2 *If $\hat{\phi}$ is a passive subset of $\hat{\theta}$, and $\hat{\theta}' \leq \hat{\theta}$, then there exists a passive subset $\hat{\phi}'$ of $\hat{\theta}'$ such that $\hat{\phi}' \leq \hat{\phi}$.*



Proof: Take $\hat{\phi}'$ to be the set of passive members of $\hat{\theta}'$. (*End of Proof*)

Now suppose two phrases p_1 and p_2 occur in some context, such as $p_1 \parallel p_2$, that prohibits their interference. Then if some identifier is used actively in either one of p_1 or p_2, it must not be used at all in the other (though it may occur in a vacuous context such as an argument to a constant procedure, or an object field that is never selected). To formalize this constraint, let $\hat{\theta}_1$ and $\hat{\theta}_2$ be the sets of types with which the identifier is used in p_1 and p_2 respectively. Then we require that $\hat{\theta}_1 \perp \hat{\theta}_2$ hold, where $\perp$ is defined to be the symmetric relation on finite sets of phrase types such that

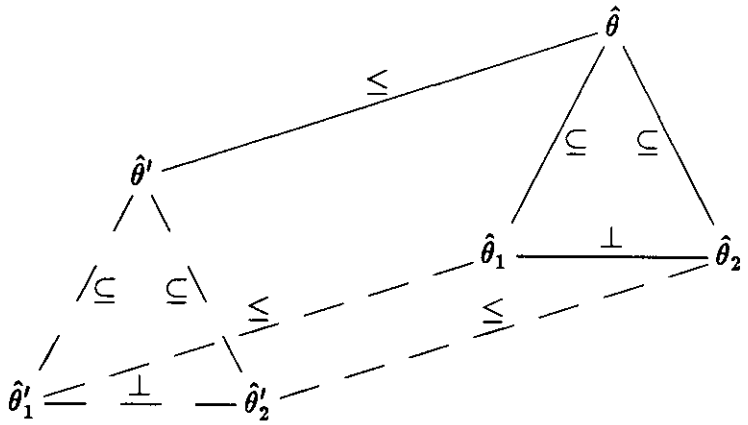
$$\hat{\theta}_1 \perp \hat{\theta}_2 \quad \text{if and only if} \quad \hat{\theta}_1 = \{\} \text{ or } \hat{\theta}_2 = \{\} \text{ or } \hat{\theta}_1 \cup \hat{\theta}_2 \text{ is passive}.$$

When $\hat{\theta}_1 \perp \hat{\theta}_2$ we say that $\hat{\theta}_1$ and $\hat{\theta}_2$ are *independent*. This relation satisfies:

Proposition 3 *If $\hat{\theta}_1 \perp \hat{\theta}_2$ and $\hat{\theta}'_1 \subseteq \hat{\theta}_1$ then $\hat{\theta}'_1 \perp \hat{\theta}_2$.*

Proposition 4 *If $\hat{\theta}_1 \perp \hat{\theta}_2$ and $\hat{\theta}'_1 \perp \hat{\theta}_2$ then $(\hat{\theta}_1 \cup \hat{\theta}'_1) \perp \hat{\theta}_2$.*

Proposition 5 *If $\hat{\theta}_1$ and $\hat{\theta}_2$ are subsets of $\hat{\theta}$ such that $\hat{\theta}_1 \perp \hat{\theta}_2$, and $\hat{\theta}' \leq \hat{\theta}$, then there exist subsets $\hat{\theta}'_1$ and $\hat{\theta}'_2$ of $\hat{\theta}'$ such that $\hat{\theta}'_1 \perp \hat{\theta}'_2$, $\hat{\theta}'_1 \leq \hat{\theta}_1$, and $\hat{\theta}'_2 \leq \hat{\theta}_2$.*



Proof: If $\hat{\theta}_1$ is empty, take $\hat{\theta}'_1$ to be empty and $\hat{\theta}'_2$ to be $\hat{\theta}'$. If $\hat{\theta}_2$ is empty, take $\hat{\theta}'_2$ to be empty and $\hat{\theta}'_1$ to be $\hat{\theta}'$. If $\hat{\theta}_1$ and $\hat{\theta}_2$ are passive, take $\hat{\phi}$ to be $\hat{\theta}_1 \cup \hat{\theta}_2$, use Proposition 2, and then take both $\hat{\theta}'_1$ and $\hat{\theta}'_2$ to be $\hat{\phi}'$. (*End of Proof*)

Next we extend the operation $\dot{\sqcup}$ from data types to phrase types. (Eventually, we will need this operation to typecheck conditional constructs.) Specifically, we define $\dot{\sqcup}$ to map pairs of types into finite sets of types as follows:

$$\delta_1\ \mathbf{exp} \mathbin{\dot{\sqcup}} \delta_2\ \mathbf{exp} = \{\, \delta\ \mathbf{exp} \mid \delta \in \delta_1 \mathbin{\dot{\sqcup}} \delta_2 \,\}$$

$$\delta_1\ \mathbf{acc} \mathbin{\dot{\sqcup}} \delta_2\ \mathbf{acc} = \{\, \delta\ \mathbf{acc} \mid \delta \in \delta_1 \mathbin{\dot{\sqcap}} \delta_2 \,\}$$

$$\mathbf{comm} \mathbin{\dot{\sqcup}} \mathbf{comm} = \{\mathbf{comm}\}$$

$$(\hat{\theta}_1 \xrightarrow[P]{} \theta_1) \mathbin{\dot{\sqcup}} (\hat{\theta}_2 \xrightarrow[P]{} \theta_2) = \{\, (\hat{\theta}_1 \cup \hat{\theta}_2) \xrightarrow[P]{} \theta \mid \theta \in \theta_1 \mathbin{\dot{\sqcup}} \theta_2 \,\}$$

$$\left.\begin{array}{l} (\hat{\theta}_1 \xrightarrow[P]{} \theta_1) \mathbin{\dot{\sqcup}} (\hat{\theta}_2 \longrightarrow \theta_2) \\ (\hat{\theta}_1 \longrightarrow \theta_1) \mathbin{\dot{\sqcup}} (\hat{\theta}_2 \xrightarrow[P]{} \theta_2) \\ (\hat{\theta}_1 \longrightarrow \theta_1) \mathbin{\dot{\sqcup}} (\hat{\theta}_2 \longrightarrow \theta_2) \end{array}\right\} = \{\, (\hat{\theta}_1 \cup \hat{\theta}_2) \longrightarrow \theta \mid \theta \in \theta_1 \mathbin{\dot{\sqcup}} \theta_2 \,\}$$

$$\iota\!:\theta_1 \mathbin{\dot{\sqcup}} \iota\!:\theta_2 = \{\, \iota\!:\theta \mid \theta \in \theta_1 \mathbin{\dot{\sqcup}} \theta_2 \,\},$$

and in any other case $\theta_1 \mathbin{\dot{\sqcup}} \theta_2 = \{\}$. Then

Proposition 6 *$\theta_1 \mathbin{\dot{\sqcup}} \theta_2$ is a finite complete set of upper bounds of θ_1 and θ_2, i.e.*

(a) *If $\theta \in \theta_1 \mathbin{\dot{\sqcup}} \theta_2$ then $\theta_1 \leq \theta$ and $\theta_2 \leq \theta$.*

(b) *If $\theta_1 \leq \theta$ and $\theta_2 \leq \theta$ then there is a $\theta_0 \in \theta_1 \mathbin{\dot{\sqcup}} \theta_2$ such that $\theta_0 \leq \theta$.*

Proof: Each half of the proposition is proved separately by induction on the structure of θ. (*End of Proof*)

(Using Propositions 1 and 6a, the reader may verify that the members of $\theta_1 \mathbin{\dot{\sqcup}} \theta_2$ never violate the requirement that, when a procedural type has a passive result type, both the procedural type and its argument type must be passive.)

Moreover, we can define the operation $\sqcup$, mapping pairs of finite sets of types into finite sets of types, such that

$$\hat{\theta}_1 \sqcup \hat{\theta}_2 = \bigcup \{\, \theta_1 \mathbin{\dot{\sqcup}} \theta_2 \mid \theta_1 \in \hat{\theta}_1 \text{ and } \theta_2 \in \hat{\theta}_2 \,\}.$$

Then

Proposition 7 *$\hat{\theta}_1 \sqcup \hat{\theta}_2$ is a least upper bound of $\hat{\theta}_1$ and $\hat{\theta}_2$.*

Now we define a *type assignment* to be a function from the set of identifiers to the set of finite sets of phrase types that maps all but a finite number of identifiers into the empty set. We say that a type assignment is *passive* when it maps every identifier into a passive set, and we use the following metavariables for type assignments:

Φ: passive type assignments

Θ: arbitrary type assignments .

We write $[\,]$ for the type assignment that maps every identifier into the empty set, and $[\,\Theta \mid \iota{:}\hat{\theta}\,]$ for the type assignment such that $[\,\Theta \mid \iota{:}\hat{\theta}\,]\iota = \hat{\theta}$ and $[\,\Theta \mid \iota{:}\hat{\theta}\,]\iota' = \Theta\iota'$ when $\iota' \neq \iota$. We also write $[\,\iota_1{:}\hat{\theta}_1 \mid \ldots \mid \iota_n{:}\hat{\theta}_n\,]$ to abbreviate $[\ldots[\,[\,] \mid \iota_1{:}\hat{\theta}_1\,]\ldots \mid \iota_n{:}\hat{\theta}_n\,]$.

We define the relations $\subseteq$, $\leq$, $\perp$, and the operation $\cup$ on type assignments by pointwise extension:

$$\Theta \subseteq \Theta' \quad \stackrel{\text{def}}{=} \quad (\forall\iota)\ \Theta\iota \subseteq \Theta'\iota$$

$$\Theta \leq \Theta' \quad \stackrel{\text{def}}{=} \quad (\forall\iota)\ \Theta\iota \leq \Theta'\iota$$

$$\Theta \perp \Theta' \quad \stackrel{\text{def}}{=} \quad (\forall\iota)\ \Theta\iota \perp \Theta'\iota$$

$$(\Theta \cup \Theta')\iota \quad \stackrel{\text{def}}{=} \quad \Theta\iota \cup \Theta'\iota\,.$$

As a consequence, Propositions 2 to 5 can be extended from finite sets of types to type assignments:

Proposition 8 *If Φ is a passive subset (in the pointwise-extended sense) of Θ, and $\Theta' \leq \Theta$, then there exists a passive subset Φ' of Θ' such that $\Phi' \leq \Phi$.*

Proposition 9 *If $\Theta_1 \perp \Theta_2$ and $\Theta_1' \subseteq \Theta_1$ then $\Theta_1' \perp \Theta_2$.*

Proposition 10 *If $\Theta_1 \perp \Theta_2$ and $\Theta_1' \perp \Theta_2$ then $(\Theta_1 \cup \Theta_1') \perp \Theta_2$.*

Proposition 11 *If Θ_1 and Θ_2 are subsets of Θ such that $\Theta_1 \perp \Theta_2$, and $\Theta' \leq \Theta$, then there exist subsets Θ_1' and Θ_2' of Θ' such that $\Theta_1' \perp \Theta_2'$, $\Theta_1' \leq \Theta_1$, and $\Theta_2' \leq \Theta_2$.*

4. Typings and their Inference Rules

If Θ is a type assignment, p is a phrase, and θ is a type, then the formula $\Theta \vdash p : \theta$, called a *typing*, asserts that the phrase p has the type θ when its free identifiers are assigned types by Θ. When $\hat{\theta}$ is a finite set of types, we write $\Theta \vdash p : \hat{\theta}$ to abbreviate the finite set of typings

$$\{\,\Theta \vdash p : \theta \mid \theta \in \hat{\theta}\,\}\,.$$

The valid typings of our illustrative language are those that are provable from the following rules of inference:

- Identifiers

$$\frac{}{\Theta \vdash \iota : \theta} \quad \text{when } \theta \in \Theta\iota$$

- Subtypes

$$\frac{\Theta \vdash p : \theta}{\Theta \vdash p : \theta'} \quad \text{when } \theta \leq \theta'$$

- Abstraction

$$\frac{[\,\Phi \mid \iota{:}\hat{\theta}\,] \vdash p : \theta}{\Theta \vdash (\lambda\iota{:}\hat{\theta}_0.\ p) : \hat{\theta} \xrightarrow[P]{} \theta} \quad \begin{array}{l}\text{when } \hat{\theta} \subseteq \hat{\theta}_0,\ \Phi \subseteq \Theta,\ \Phi \text{ is passive,} \\ \text{and if } \theta \text{ is passive then } \hat{\theta} \text{ is passive}\end{array}$$

$$\frac{[\,\Theta \mid \iota{:}\hat{\theta}\,] \vdash p : \alpha}{\Theta \vdash (\lambda\iota{:}\hat{\theta}_0.\ p) : \hat{\theta} \longrightarrow \alpha} \quad \text{when } \hat{\theta} \subseteq \hat{\theta}_0$$

- Application

$$\frac{\begin{array}{l}\Theta_1 \vdash p_1 : \hat{\theta} \xrightarrow[P]{} \theta \\ \Theta_2 \vdash p_2 : \hat{\theta}\end{array}}{\Theta \vdash p_1\, p_2 : \theta} \quad \begin{array}{l}\text{when } \Theta_1 \subseteq \Theta,\ \Theta_2 \subseteq \Theta,\ \Theta_1 \perp \Theta_2 \\ \text{and if } \theta \text{ is passive then } \hat{\theta} \text{ is passive}\end{array}$$

$$\frac{\begin{array}{l}\Theta_1 \vdash p_1 : \hat{\theta} \longrightarrow \alpha \\ \Theta_2 \vdash p_2 : \hat{\theta}\end{array}}{\Theta \vdash p_1\, p_2 : \alpha} \quad \text{when } \Theta_1 \subseteq \Theta,\ \Theta_2 \subseteq \Theta, \text{ and } \Theta_1 \perp \Theta_2$$

- Tupling

$$\frac{\Theta \vdash p_k : \theta}{\Theta \vdash \langle \iota_1 \equiv p_1,\ \ldots\ ,\ \iota_k \equiv p_k,\ \ldots\ ,\ \iota_n \equiv p_n \rangle : (\iota_k{:}\theta)}$$

- Field Selection

$$\frac{\Theta \vdash p : (\iota{:}\theta)}{\Theta \vdash p.\iota : \theta}$$

- Conditionals

$$\frac{\begin{array}{l}\Theta \vdash p_1 : \textbf{bool exp} \\ \Theta \vdash p_2 : \theta \\ \Theta \vdash p_3 : \theta\end{array}}{\Theta \vdash \textbf{if } p_1 \textbf{ then } p_2 \textbf{ else } p_3 : \theta}$$

- Arithmetic Expressions

$$\frac{}{\Theta \vdash 0 : \textbf{int exp}} \qquad \frac{}{\Theta \vdash 0.5 : \textbf{real exp}}$$

$$\frac{\begin{array}{l}\Theta \vdash p_1 : \textbf{int exp} \\ \Theta \vdash p_2 : \textbf{int exp}\end{array}}{\Theta \vdash p_1 + p_2 : \textbf{int exp}} \qquad \frac{\begin{array}{l}\Theta \vdash p_1 : \textbf{real exp} \\ \Theta \vdash p_2 : \textbf{real exp}\end{array}}{\Theta \vdash p_1 + p_2 : \textbf{real exp}}$$

- Commands

$$\frac{\Theta \vdash p_1 : \delta\ \mathbf{acc} \qquad \Theta \vdash p_2 : \delta\ \mathbf{exp}}{\Theta \vdash p_1 := p_2 : \mathbf{comm}}$$

$$\frac{\Theta \vdash p_1 : \mathbf{comm} \qquad \Theta \vdash p_2 : \mathbf{comm}}{\Theta \vdash p_1 ; p_2 : \mathbf{comm}} \qquad \frac{\Theta \vdash p_1 : \mathbf{bool\ exp} \qquad \Theta \vdash p_2 : \mathbf{comm}}{\Theta \vdash \mathbf{while}\ p_1\ \mathbf{do}\ p_2 : \mathbf{comm}}$$

- Concurrency

$$\frac{\Theta_1 \vdash p_1 : \mathbf{comm} \qquad \Theta_2 \vdash p_2 : \mathbf{comm}}{\Theta \vdash p_1 \parallel p_2 : \mathbf{comm}} \quad \text{when } \Theta_1 \subseteq \Theta,\ \Theta_2 \subseteq \Theta, \text{ and } \Theta_1 \perp \Theta_2$$

In the rules for an application $p_1 p_2$, notice that the requirement $\Theta_1 \perp \Theta_2$ prohibits interference between p_1 and p_2 (just as with the concurrent construction $p_1 \parallel p_2$), so that a procedure must not interfere with its argument. This is the basic mechanism that insures that reduction preserves syntactic correctness.

In the rules for abstraction, the condition $\hat{\theta} \subseteq \hat{\theta}_0$ (where $\hat{\theta}_0$ is the finite set of types occurring explicitly in the lambda expression) restricts the procedural type that can be inferred; this restriction is introduced to make typechecking feasible.

In what follows, we will prove several propositions by induction on the size of a proof, using the above rules, of a typing, with a case analysis over the different inference rules that may occur at the root of the proof tree. Fortunately, most of the inference rules fall into one of two classes that can be treated uniformly in such a case analysis:

- An inference rule is called a *normal rule* if it is equivalent to a (possibly infinite) set of rules of the form

$$\frac{\begin{array}{l}\Theta \vdash p_1 : \hat{\theta}_1 \\ \quad\vdots \\ \Theta \vdash p_n : \hat{\theta}_n\end{array}}{\Theta \vdash \varepsilon(p_1, \ldots, p_n) : \theta}$$

  where Θ, $p_1, \ldots, p_n$ are metavariables, $\varepsilon(p_1, \ldots, p_n)$ is a phrase constructed from $p_1, \ldots, p_n$ without using binding operations, $\hat{\theta}_1, \ldots, \hat{\theta}_n$ are finite sets of types not containing metavariables, θ is a type not containing metavariables, and if θ is passive then $\hat{\theta}_1, \ldots, \hat{\theta}_n$ are passive.

- An inference rule is called a *noninterference rule* if it is equivalent to a (possibly infinite) set of rules of the form

$$\frac{\Theta_1 \vdash p_1 : \hat{\theta}_1 \quad \Theta_2 \vdash p_2 : \hat{\theta}_2}{\Theta \vdash \varepsilon(p_1, p_2) : \theta} \quad \text{when } \Theta_1 \subseteq \Theta,\ \Theta_2 \subseteq \Theta, \text{ and } \Theta_1 \perp \Theta_2$$

  where Θ_1, Θ_2, Θ, p_1, and p_2 are metavariables, $\varepsilon(p_1, p_2)$ is a phrase constructed from p_1 and p_2 without using binding operations, $\hat{\theta}_1$ and $\hat{\theta}_2$ are finite sets of types not containing metavariables, θ is a type not containing metavariables, and if θ is passive then $\hat{\theta}_1$ and $\hat{\theta}_2$ are passive.

For example, when k and n are integers such that $1 \leq k \leq n$, θ is a type, and $\iota_1, \ldots, \iota_n$ are identifiers, let $\mathcal{R}_{nk\theta\iota_1\ldots\iota_n}$ be the rule

$$\frac{\Theta \vdash p_k : \{\theta\}}{\Theta \vdash \langle \iota_1 \equiv p_1, \ \ldots\ , \iota_k \equiv p_k, \ \ldots\ , \iota_n \equiv p_n \rangle : (\iota_k : \theta)}$$

Then the inference rule for object construction is equivalent to the set of rules

$$\{\, \mathcal{R}_{nk\theta\iota_1\ldots\iota_n} \mid 1 \leq k \leq n \text{ and } \theta \text{ is a type and } \iota_1, \ldots, \iota_n \text{ are identifiers} \,\},$$

and is therefore a normal rule.

On the other hand, when $\hat{\theta}$ is a finite set of types and α is an active type, let $\mathcal{R}_{\hat{\theta}\alpha}$ be the rule

$$\frac{\Theta_1 \vdash p_1 : \{\hat{\theta} \longrightarrow \alpha\} \quad \Theta_2 \vdash p_2 : \hat{\theta}}{\Theta \vdash p_1\, p_2 : \alpha} \quad \text{when } \Theta_1 \subseteq \Theta,\ \Theta_2 \subseteq \Theta, \text{ and } \Theta_1 \perp \Theta_2$$

Then the second inference rule for application is equivalent to the set of rules

$$\{\, \mathcal{R}_{\hat{\theta}\alpha} \mid \hat{\theta} \text{ is a finite set of types and } \alpha \text{ is a type} \,\},$$

and is therefore a noninterference rule.

The reader may verify that, except for the rule for identifiers and the two rules for abstraction, every inference rule is either a normal rule or a noninterference rule.

Proposition 12 *If* $\Theta' \leq \Theta$ *and* $\Theta \vdash p : \theta$ *then* $\Theta' \vdash p : \theta$.

Proof: By induction on the proof size of $\Theta \vdash p : \theta$.

(1) If the proof root is the rule for identifiers, then p is an identifier ι and $\theta \in \Theta\iota$. Since $\Theta'\iota \leq \Theta\iota$, there is a $\theta' \in \Theta'\iota$ such that $\theta' \leq \theta$. Then the identifier rule gives $\Theta' \vdash \iota : \theta'$ and the subtype rule gives $\Theta' \vdash \iota : \theta$.

(2) If the proof root is a normal rule, then p must have the form $\varepsilon(p_1, \ldots, p_n)$, and the premisses of the rule must have the forms $\Theta \vdash p_1 : \hat{\theta}_1, \ldots, \Theta \vdash p_n : \hat{\theta}_n$. By the induction hypothesis (applied to each member of each $\hat{\theta}_i$), $\Theta' \vdash p_1 : \hat{\theta}_1, \ldots, \Theta' \vdash p_n : \hat{\theta}_n$, and by the rule used at the root, $\Theta' \vdash \varepsilon(p_1, \ldots, p_n) : \theta$.

(3) If the proof root is a noninterference rule, then p must have the form $\varepsilon(p_1, p_2)$ and the premisses of the rule must have the forms $\Theta_1 \vdash p_1 : \hat{\theta}_1$ and $\Theta_2 \vdash p_2 : \hat{\theta}_2$, where Θ_1 and Θ_2 are subsets of Θ such that $\Theta_1 \perp \Theta_2$. By Proposition 11, there are subsets Θ'_1 and Θ'_2 of Θ' such that $\Theta'_1 \perp \Theta'_2$, $\Theta'_1 \leq \Theta_1$, and $\Theta'_2 \leq \Theta_2$. By the induction hypothesis, $\Theta'_1 \vdash p_1 : \hat{\theta}_1$ and $\Theta'_2 \vdash p_2 : \hat{\theta}_2$, and by the rule used at the root, $\Theta' \vdash \varepsilon(p_1, p_2) : \theta$.

(4) If the proof root is the first rule for abstraction, then its premiss has the form $[\,\Phi \mid \iota{:}\hat{\theta}\,] \vdash \cdots$, where Φ is a passive subset of Θ. By Proposition 8, there is a passive subset Φ' of Θ' such that $\Phi' \leq \Phi$. Then $[\,\Phi' \mid \iota{:}\hat{\theta}\,] \leq [\,\Phi \mid \iota{:}\hat{\theta}\,]$, and thus the induction hypothesis allows us to replace $[\,\Phi \mid \iota{:}\hat{\theta}\,]$ by $[\,\Phi' \mid \iota{:}\hat{\theta}\,]$, and the rule at the root allows us to replace Θ by Θ'.

The more straightforward case where the proof root is the second rule for abstraction is left to the reader. (*End of Proof*)

Proposition 13 (a) *If ϕ is passive and $\Theta \vdash p : \phi$ then there exists a passive $\Phi \subseteq \Theta$ such that $\Phi \vdash p : \phi$.*

(b) *If $\hat{\phi}$ is passive and $\Theta \vdash p : \hat{\phi}$ then there exists a passive $\Phi \subseteq \Theta$ such that $\Phi \vdash p : \hat{\phi}$.*

Proof: By induction on the proof size of $\Phi \vdash p : \phi$ or $\Phi \vdash p : \hat{\phi}$. Within the induction step, we prove (a) by case analysis of the proof root and then show that (b) follows from (a).

(a1) If the proof root is the rule for identifiers, then p is an identifier ι and $\phi \in \Theta\iota$, so that one can take Φ to be $[\,\iota{:}\{\phi\}\,]$.

(a2) If the proof root is a normal rule, then p must have the form $\varepsilon(p_1, \ldots, p_n)$ and the premisses of the rule must have the forms $\Theta \vdash p_1 : \hat{\phi}_1, \ldots, \Theta \vdash p_n : \hat{\phi}_n$, where the $\hat{\phi}_i$'s are passive. By the induction hypothesis there are passive $\Phi_1, \ldots, \Phi_n \subseteq \Theta$ such that $\Phi_1 \vdash p_1 : \hat{\phi}_1, \ldots, \Phi_n \vdash p_n : \hat{\phi}_n$. Let Φ be $\Phi_1 \cup \cdots \cup \Phi_n$, which is a passive subset of Θ. Then, since $\Phi_i \subseteq \Phi$ implies $\Phi \leq \Phi_i$, Proposition 12 gives $\Phi \vdash p_1 : \hat{\phi}_1, \ldots, \Phi \vdash p_n : \hat{\phi}_n$, and the rule used at the root gives $\Phi \vdash \varepsilon(p_1, \ldots, p_n) : \phi$.

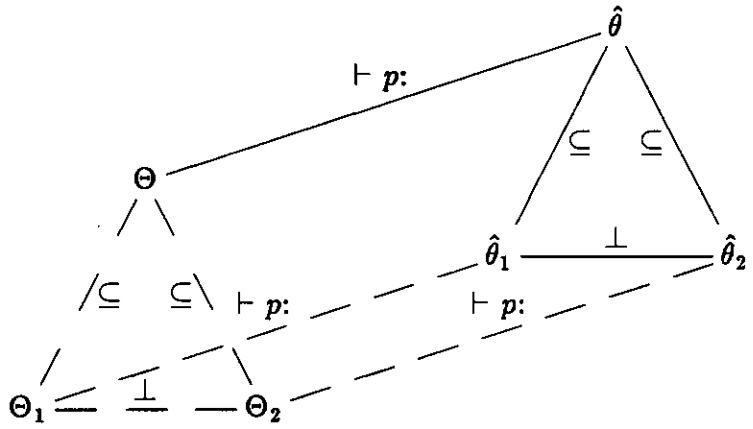
(a3) If the proof root is a noninterference rule, then p must have the form $\varepsilon(p_1, p_2)$ and the premisses of the rule must have the forms $\Theta_1 \vdash p_1 : \hat{\phi}_1$ and $\Theta_2 \vdash p_2 : \hat{\phi}_2$, where Θ_1 and Θ_2 are subsets of Θ such that $\Theta_1 \perp \Theta_2$, and $\hat{\phi}_1$ and $\hat{\phi}_2$ are passive. By the induction hypothesis there are passive $\Phi_1 \subseteq \Theta_1$ and $\Phi_2 \subseteq \Theta_2$ such that $\Phi_1 \vdash p_1 : \hat{\phi}_1$ and $\Phi_2 \vdash p_2 : \hat{\phi}_2$. Let Φ be $\Phi_1 \cup \Phi_2$, which is a passive subset of Θ. Then Φ_1 and Φ_2 are subsets of Φ that, since they are passive, satisfy $\Phi_1 \perp \Phi_2$. Thus the rule used at the root gives $\Phi \vdash \varepsilon(p_1, p_2) : \phi$.

(a4) If the proof root is the first rule for abstraction, then its premiss has the form $[\,\Phi \mid \iota{:}\hat{\theta}\,] \vdash \cdots$, where $\Phi \subseteq \Theta$ is passive. Then, since Φ is a subset of itself, we can replace Θ by Φ in the consequence of the rule.

The proof root cannot be the second rule for abstraction, since the consequence of this rule never has the form $\Theta \vdash p : \phi$ for passive ϕ.

(b) If $\Theta \vdash p : \hat{\phi}$, then $\Theta \vdash p : \phi_i$ for each of the finitely many $\phi_i \in \hat{\phi}$. From (a), for each ϕ_i there is a passive $\Phi_i \subseteq \Theta$ such that $\Phi_i \vdash p : \phi_i$. Let Φ be the union of these Φ_i, which is a passive subset of Θ. Then, since $\Phi_i \subseteq \Phi$ implies $\Phi \leq \Phi_i$, Proposition 12 gives $\Phi \vdash p : \phi_i$ for each ϕ_i, and thus $\Phi \vdash p : \hat{\phi}$. *(End of Proof)*

Proposition 14 *If $\hat{\theta}_1$ and $\hat{\theta}_2$ are subsets of $\hat{\theta}$ such that $\hat{\theta}_1 \perp \hat{\theta}_2$, and $\Theta \vdash p : \hat{\theta}$, then there exist subsets Θ_1 and Θ_2 of Θ such that such that $\Theta_1 \perp \Theta_2$, $\Theta_1 \vdash p : \hat{\theta}_1$, and $\Theta_2 \vdash p : \hat{\theta}_2$.*



Proof: If $\hat{\theta}_1$ is empty, take Θ_1 to be empty and Θ_2 to be Θ. If $\hat{\theta}_2$ is empty, take Θ_2 to be empty and Θ_1 to be Θ. If $\hat{\theta}_1$ and $\hat{\theta}_2$ are passive, take $\hat{\phi}$ to be $\hat{\theta}_1 \cup \hat{\theta}_2$, use Proposition 13b, and then take both Θ_1 and Θ_2 to be Φ. *(End of Proof)*

With these preliminaries, we can prove the basic relationship between typings and substitutions. We write $(p_1/\iota \rightarrow p_2)$ to denote the result of substituting p_2 for the free occurrences of ι in p_1, with renaming to avoid identifier collisions.

Proposition 15 *If $[\,\Theta_1 \mid \iota{:}\hat{\theta}\,] \vdash p_1 : \theta$, $\Theta_2 \vdash p_2 : \hat{\theta}$, and Θ_1 and Θ_2 are subsets of Θ such that $\Theta_1 \perp \Theta_2$, then $\Theta \vdash (p_1/\iota \rightarrow p_2) : \theta$.*

Proof: By induction on the proof size of $[\,\Theta_1 \mid \iota{:}\hat{\theta}\,] \vdash p_1 : \theta$.

(1) If the proof root is the rule for identifiers, then p_1 is an identifier ι' and $\theta \in [\,\Theta_1 \mid \iota{:}\hat{\theta}\,]\iota'$. If $\iota' = \iota$ then $\theta \in \hat{\theta}$ and $(p_1/\iota \rightarrow p_2) = p_2$, so that $\Theta_2 \vdash p_2 : \hat{\theta}$ gives $\Theta_2 \vdash (p_1/\iota \rightarrow p_2) : \theta$ and, since $\Theta_2 \subseteq \Theta$ implies $\Theta \leq \Theta_2$, Proposition 12 gives $\Theta \vdash (p_1/\iota \rightarrow p_2) : \theta$. On the

other hand, if $\iota' \neq \iota$ then $\theta \in \Theta_1 \iota'$ and $(p_1/\iota \to p_2) = \iota'$, so that $\Theta_1 \vdash (p_1/\iota \to p_2) : \theta$ and, since $\Theta_1 \subseteq \Theta$ implies $\Theta \leq \Theta_1$, Proposition 12 gives $\Theta \vdash (p_1/\iota \to p_2) : \theta$.

(2) If the proof root is a normal rule, then p_1 must have the form $\varepsilon(p_{11}, \ldots, p_{1n})$, which is constructed from $p_{11}, \ldots, p_{1n}$ without using binding operators, and the premisses of the rule must have the form

$$[\,\Theta_1 \mid \iota{:}\,\hat{\theta}\,] \vdash p_{11} : \hat{\theta}_1 \qquad \ldots \qquad [\,\Theta_1 \mid \iota{:}\,\hat{\theta}\,] \vdash p_{1n} : \hat{\theta}_n\,.$$

By the induction hypothesis (applied to each member of each $\hat{\theta}_i$),

$$\Theta \vdash (p_{11}/\iota \to p_2) : \hat{\theta}_1 \qquad \ldots \qquad \Theta \vdash (p_{1n}/\iota \to p_2) : \hat{\theta}_n\,,$$

and by the rule at the root,

$$\Theta \vdash \varepsilon((p_{11}/\iota \to p_2), \ldots, (p_{1n}/\iota \to p_2)) : \theta\,.$$

But since ε is constructed without binding operators,

$$\varepsilon((p_{11}/\iota \to p_2), \ldots, (p_{1n}/\iota \to p_2)) = (\varepsilon(p_{11}, \ldots, p_{1n})/\iota \to p_2) = (p_1/\iota \to p_2)\,.$$

(3) If the proof root is a noninterference rule, then p_1 must have the form $\varepsilon(p_{11}, p_{12})$, which is constructed from p_{11} and p_{12} without using binding operators, and the premisses of the rule must have the forms

$$[\,\Theta_{11} \mid \iota{:}\,\hat{\theta}_1\,] \vdash p_{11} : \hat{\theta}'_1 \qquad \text{and} \qquad [\,\Theta_{12} \mid \iota{:}\,\hat{\theta}_2\,] \vdash p_{12} : \hat{\theta}'_2\,,$$

where

$$\Theta_{11} \subseteq \Theta_1 \qquad \Theta_{12} \subseteq \Theta_1 \qquad \Theta_{11} \perp \Theta_{12}$$
$$\hat{\theta}_1 \subseteq \hat{\theta} \qquad \hat{\theta}_2 \subseteq \hat{\theta} \qquad \hat{\theta}_1 \perp \hat{\theta}_2\,.$$

Then, since $\Theta_2 \vdash p_2 : \hat{\theta}$, Proposition 14 shows that there are Θ_{21} and Θ_{22} such that

$$\Theta_{21} \subseteq \Theta_2 \qquad \Theta_{22} \subseteq \Theta_2 \qquad \Theta_{21} \perp \Theta_{22}$$
$$\Theta_{21} \vdash p_2 : \hat{\theta}_1 \qquad \Theta_{22} \vdash p_2 : \hat{\theta}_2\,.$$

Since $\Theta_1 \perp \Theta_2$, Proposition 9 gives

$$\Theta_{11} \perp \Theta_{21} \qquad \Theta_{12} \perp \Theta_{22} \qquad \Theta_{11} \perp \Theta_{22} \qquad \Theta_{12} \perp \Theta_{21}\,.$$

Then the induction hypothesis (applied to each member of $\hat{\theta}'_1$ and $\hat{\theta}'_2$) gives

$$\Theta_{11} \cup \Theta_{21} \vdash (p_{11}/\iota \to p_2) : \hat{\theta}'_1 \qquad \Theta_{12} \cup \Theta_{22} \vdash (p_{12}/\iota \to p_2) : \hat{\theta}'_2\,.$$

By Proposition 10, $(\Theta_{11} \cup \Theta_{21}) \perp (\Theta_{12} \cup \Theta_{22})$. Thus the rule used at the root gives

$$\Theta \vdash \varepsilon\big((p_{11}/\iota \to p_2), (p_{12}/\iota \to p_2)\big) : \theta\,,$$

or, since ε is constructed without binding operators,

$$\Theta \vdash (p_1/\iota \to p_2) : \theta \,.$$

(4) If the proof root is the first abstraction rule, then its instance must have the form

$$\frac{[\,[\,\Phi_1 \mid \iota{:}\,\hat{\phi}\,] \mid \iota'{:}\,\hat{\theta}'\,] \vdash p_1' : \theta'}{[\,\Theta_1 \mid \iota{:}\,\hat{\theta}\,] \vdash (\lambda\iota'{:}\,\hat{\theta}_0.\ p_1') : \hat{\theta}' \xrightarrow[P]{} \theta'}$$

where $\hat{\theta}' \subseteq \hat{\theta}_0$, $\Phi_1 \subseteq \Theta_1$ is passive, and $\hat{\phi} \subseteq \hat{\theta}$ is passive. Moreover, since renaming obviously preserves typings, we can assume without loss of generality that ι' is distinct from ι and does not occur free in p_2, and thus that Θ_2 maps ι' into the empty set.

Since $\hat{\phi}$ is a passive subset of $\hat{\theta}$ and $\Theta_2 \vdash p_2 : \hat{\theta}$, by Proposition 13b there is a passive $\Phi_2 \subseteq \Theta_2$ such that $\Phi_2 \vdash p_2 : \hat{\phi}$. Since $\iota' \neq \iota$, the premiss of the abstraction rule instance can be rewritten as

$$[\,[\,\Phi_1 \mid \iota'{:}\,\hat{\theta}'\,] \mid \iota{:}\,\hat{\phi}\,] \vdash p_1' : \theta' \,,$$

and since $\Phi_2\iota'$ is empty and Φ_1 and Φ_2 are passive,

$$[\,\Phi_1 \mid \iota'{:}\,\hat{\theta}'\,] \perp \Phi_2 \qquad \text{and} \qquad [\,\Phi_1 \mid \iota'{:}\,\hat{\theta}'\,] \cup \Phi_2 = [\,\Phi_1 \cup \Phi_2 \mid \iota'{:}\,\hat{\theta}'\,] \,.$$

Thus the induction hypothesis gives

$$[\,\Phi_1 \cup \Phi_2 \mid \iota'{:}\,\hat{\theta}'\,] \vdash (p_1'/\iota \to p_2) : \theta' \,,$$

and, since $\Phi_1 \cup \Phi_2$ is a passive subset of Θ, the first abstraction rule gives

$$\Theta \vdash \left(\lambda\iota'{:}\,\hat{\theta}_0.\ (p_1'/\iota \to p_2)\right) : \hat{\theta}' \xrightarrow[P]{} \theta' \,.$$

Finally, since ι' is distinct from ι and does not occur free in p_2,

$$((\lambda\iota'{:}\,\hat{\theta}_0.\ p_1')/\iota \to p_2) = \lambda\iota'{:}\,\hat{\theta}_0.\ (p_1'/\iota \to p_2) \,.$$

The simpler case of the second abstraction rule is left to the reader. (*End of Proof*)

Notice, however, that the converse of Proposition 15 does not hold. For example, suppose

p_1	is	$(x.a)(x.b)$
p_2	is	$\langle a \equiv y, b \equiv z\rangle$
ι	is	x
Θ	is	$[\,y{:}\,\{\{\mathbf{comm}\} \longrightarrow \mathbf{comm}\} \mid z{:}\,\{\mathbf{comm}\}\,]$
θ	is	$\mathbf{comm}$

Then $\Theta \vdash (p_1/\iota \to p_2) : \theta$ is

$$[\,y{:}\,\{\{\mathbf{comm}\} \longrightarrow \mathbf{comm}\} \mid z{:}\,\{\mathbf{comm}\}\,] \vdash (\langle a \equiv y, b \equiv z\rangle.a)(\langle a \equiv y, b \equiv z\rangle.b) : \mathbf{comm} \,,$$

which is a valid typing. However, there are no $\Theta_1, \Theta_2 \subseteq \Theta$ and $\hat{\theta}$ such that

$$[\,\Theta_1 \mid x{:}\,\hat{\theta}\,] \vdash (x.a)(x.b) : \mathbf{comm}$$

$$\Theta_2 \vdash \langle a \equiv y, b \equiv z\rangle : \hat{\theta}\,,$$

since the second typing requires every member of $\hat{\theta}$ to be either $b{:}\,\mathbf{comm}$ or $a{:}\,\hat{\theta}' \longrightarrow \mathbf{comm}$ where $\mathbf{comm} \in \hat{\theta}'$, which makes the first typing impossible.

From Proposition 15, the reader may verify that beta reduction preserves typings:

Proposition 16 *If* $\Theta \vdash (\lambda\iota{:}\,\hat{\theta}_0.\ p_1)p_2 : \theta$ *then* $\Theta \vdash (p_1/\iota \rightarrow p_2) : \theta$.

The reader may also verify that the reduction of tuples, and its inverse, preserves typings:

Proposition 17 $\Theta \vdash \langle \iota_1 \equiv p_1,\ \ldots\ , \iota_k \equiv p_k,\ \ldots\ , \iota_n \equiv p_n\rangle.\iota_k : \theta$ *iff* $\Theta \vdash p_k : \theta$.

5. Typechecking

To show that the typings defined in the previous section can be checked, we define a computable typechecking function Ψ, which accepts a type assignment and a phrase. Essentially $\Psi(\Theta_0, p)$ produces a finite set $\hat{\theta}$ of types such that $\Theta_0 \vdash p : \theta$ holds if and only if there is a member of $\hat{\theta}$ that is a subtype of θ. However, $\Psi(\Theta_0, p)$ also produces additional information: the $\theta \in \hat{\theta}$ are paired with type assignments $\Theta \subseteq \Theta_0$ that are just sufficient to give $\Theta \vdash p : \theta$. Thus $\Psi(\Theta_0, p)$ is a set of pairs, each consisting of a type assignment and a type. This function is defined by induction on the structure of phrases:

$$\Psi(\Theta_0, \iota) = \left\{ \langle [\,\iota{:}\,\{\theta\}\,], \theta\rangle \;\middle|\; \theta \in \Theta_0\iota \right\}$$

$$\Psi(\Theta_0, \lambda\iota{:}\,\hat{\theta}_0.\ p) =$$
$$\left\{ \langle \Theta, \hat{\theta} \xrightarrow[P]{} \theta\rangle \;\middle|\; \langle [\,\Theta \mid \iota{:}\,\hat{\theta}\,], \theta\rangle \in \Psi([\,\Theta_0 \mid \iota{:}\,\hat{\theta}_0\,], p) \text{ and } \Theta\iota = \{\} \text{ and } \Theta \text{ passive} \right\} \cup$$
$$\left\{ \langle \Theta, \hat{\theta} \longrightarrow \theta\rangle \;\middle|\; \langle [\,\Theta \mid \iota{:}\,\hat{\theta}\,], \theta\rangle \in \Psi([\,\Theta_0 \mid \iota{:}\,\hat{\theta}_0\,], p) \text{ and } \Theta\iota = \{\} \text{ and } \Theta \text{ not passive} \right\}$$

$$\Psi(\Theta_0, p_1p_2) = \left\{ \langle \Theta_1 \cup \Theta_2, \theta\rangle \;\middle|\; \Theta_1 \perp \Theta_2 \text{ and} \right.$$
$$(\exists\hat{\theta})\ \Big(\big(\langle \Theta_1, \hat{\theta} \xrightarrow[P]{} \theta\rangle \in \Psi(\Theta_0, p_1) \text{ or } \langle \Theta_1, \hat{\theta} \longrightarrow \theta\rangle \in \Psi(\Theta_0, p_1)\big) \text{ and}$$
$$\left.\left.(\exists\eta \in \hat{\theta} \Rightarrow \Psi(\Theta_0, p_2))\ \Big(\Theta_2 = \bigcup_{\theta' \in \hat{\theta}} [\eta\theta']_1 \text{ and } (\forall\theta' \in \hat{\theta})\ [\eta\theta']_2 \leq \theta'\Big)\right)\right\}$$

$$\Psi(\Theta_0, \langle \iota_1 \equiv p_1,\ \ldots\ , \iota_n \equiv p_n\rangle) = \bigcup_{k=1}^{n} \left\{ \langle \Theta, \iota_k{:}\,\theta\rangle \;\middle|\; \langle \Theta, \theta\rangle \in \Psi(\Theta_0, p_k) \right\}$$

$$\Psi(\Theta_0, p.\iota) = \left\{ \langle \Theta, \theta\rangle \;\middle|\; \langle \Theta, \iota{:}\,\theta\rangle \in \Psi(\Theta_0, p) \right\}$$

$$\Psi(\Theta_0, \textbf{if } p_1 \textbf{ then } p_2 \textbf{ else } p_3) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2 \cup \Theta_3, \theta \rangle \;\Big|\; (\exists \theta_1, \theta_2, \theta_3)\ \theta_1 \leq \textbf{bool exp} \text{ and } \theta \in \theta_2 \dot{\sqcup} \theta_3 \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \text{ and } \langle \Theta_3, \theta_3 \rangle \in \Psi(\Theta_0, p_3) \Big\}$$

$$\Psi(\Theta_0, 0) = \Big\{ \langle [\,], \textbf{int exp} \rangle \Big\}$$

$$\Psi(\Theta_0, 0.5) = \Big\{ \langle [\,], \textbf{real exp} \rangle \Big\}$$

$$\Psi(\Theta_0, p_1 + p_2) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{int exp} \rangle \;\Big|\; (\exists \theta_1, \theta_2)\ \theta_1 \leq \textbf{int exp} \text{ and } \theta_2 \leq \textbf{int exp} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\} \cup$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{real exp} \rangle \;\Big|\; (\exists \theta_1, \theta_2)\ \theta_1 \leq \textbf{real exp} \text{ and } \theta_2 \leq \textbf{real exp} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\}$$

$$\Psi(\Theta_0, p_1 := p_2) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{comm} \rangle \;\Big|\; (\exists \theta_1, \theta_2, \delta)\ \theta_1 \leq \delta\ \textbf{acc} \text{ and } \theta_2 \leq \delta\ \textbf{exp} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\}$$

$$\Psi(\Theta_0, p_1 \,;\, p_2) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{comm} \rangle \;\Big|\; (\exists \theta_1, \theta_2)\ \theta_1 \leq \textbf{comm} \text{ and } \theta_2 \leq \textbf{comm} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\}$$

$$\Psi(\Theta_0, \textbf{while } p_1 \textbf{ do } p_2) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{comm} \rangle \;\Big|\; (\exists \theta_1, \theta_2)\ \theta_1 \leq \textbf{bool exp} \text{ and } \theta_2 \leq \textbf{comm} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\}$$

$$\Psi(\Theta_0, p_1 \parallel p_2) =$$
$$\Big\{ \langle \Theta_1 \cup \Theta_2, \textbf{comm} \rangle \;\Big|\; \Theta_1 \perp \Theta_2 \text{ and } (\exists \theta_1, \theta_2)\ \theta_1 \leq \textbf{comm} \text{ and } \theta_2 \leq \textbf{comm} \text{ and } \langle \Theta_1, \theta_1 \rangle \in \Psi(\Theta_0, p_1) \text{ and } \langle \Theta_2, \theta_2 \rangle \in \Psi(\Theta_0, p_2) \Big\}$$

In the equation for $\Psi(\Theta_0, p_1 p_2)$, the expression $\hat{\theta} \Rightarrow \Psi(\Theta_0, p_2)$ denotes the finite set of all functions from $\hat{\theta}$ to $\Psi(\Theta_0, p_2)$, and $[\eta\theta']_1$ and $[\eta\theta']_2$ denote the first and second components of the pair $\eta\theta'$.

This typechecking function meets the following specification:

Proposition 18 (a) *If* $\langle \Theta, \theta \rangle \in \Psi(\Theta_0, p)$ *then* $\Theta \subseteq \Theta_0$ *and* $\Theta \vdash p : \theta$ *and if* θ *is passive then* Θ *is passive.*

(b) *If* $\Theta \subseteq \Theta_0$ *and* $\Theta \vdash p : \theta$ *then* $(\exists \langle \Theta', \theta' \rangle \in \Psi(\Theta_0, p))\ \Theta' \subseteq \Theta$ *and* $\theta' \leq \theta$.

Proof: (a) By induction on the structure of p. We give the details of the case where p is an application p_1p_2, and leave the tedium of the remaining cases to the reader.

Suppose $\langle \Theta, \theta \rangle \in \Psi(\Theta_0, p_1p_2)$. By the definition of Ψ, there are Θ_1, Θ_2, and $\hat{\theta}$ such that $\Theta = \Theta_1 \cup \Theta_2$, $\Theta_1 \perp \Theta_2$, and either $\langle \Theta_1, \hat{\theta} \xrightarrow[P]{} \theta \rangle$ or $\langle \Theta_1, \hat{\theta} \longrightarrow \theta \rangle$ belongs to $\Psi(\Theta_0, p_1)$. Also, there is a function η from $\hat{\theta}$ to $\Psi(\Theta_0, p_2)$ such that

$$\Theta_2 = \bigcup_{\theta' \in \hat{\theta}} [\eta\theta']_1 \qquad \text{and} \qquad (\forall \theta' \in \hat{\theta})\ [\eta\theta']_2 \leq \theta' .$$

For each $\theta' \in \hat{\theta}$, since $\eta\theta' \in \Psi(\Theta_0, p_2)$, the induction hypothesis for p_2 gives $[\eta\theta']_1 \subseteq \Theta_0$ and $[\eta\theta']_1 \vdash p_2 : [\eta\theta']_2$ and if $[\eta\theta']_2$ is passive then $[\eta\theta']_1$ is passive. Then, since $[\eta\theta']_2 \leq \theta'$, we have $[\eta\theta']_1 \vdash p_2 : \theta'$ and, by Proposition 1, if θ' is passive then $[\eta\theta']_1$ is passive. Thus, since Θ_2 is the union of $[\eta\theta']_1$ over $\theta' \in \hat{\theta}$, we have $\Theta_2 \subseteq \Theta_0$ and $\Theta_2 \vdash p_2 : \hat{\theta}$ and if $\hat{\theta}$ is passive then Θ_2 is passive.

By the induction hypothesis for p_1, we have $\Theta_1 \subseteq \Theta_0$ and either $\Theta_1 \vdash p_1 : \hat{\theta} \xrightarrow[P]{} \theta$ and Θ_1 is passive (since $\hat{\theta} \xrightarrow[P]{} \theta$ is passive) or $\Theta_1 \vdash p_1 : \hat{\theta} \longrightarrow \theta$. Thus $\Theta = \Theta_1 \cup \Theta_2 \subseteq \Theta_0$ and, by the inference rules for application, $\Theta \vdash p_1p_2 : \theta$. Moreover, if θ is passive then our restrictions on procedural types prohibit the type $\hat{\theta} \longrightarrow \theta$ and insure that $\hat{\theta}$ is passive, so that Θ_1 and Θ_2, and thus Θ, are passive.

(b) By induction on the proof size of $\Theta \vdash p : \theta$. We give the details of the case where the proof root is an application rule, and leave the remaining cases to the reader.

Suppose $\Theta \subseteq \Theta_0$ and the root of the proof of $\Theta \vdash p : \theta$ is one of the inference rules for application. Then $p = p_1p_2$, either $\Theta_1 \vdash p_1 : \hat{\theta} \xrightarrow[P]{} \theta$ or $\Theta_1 \vdash p_1 : \hat{\theta} \longrightarrow \theta$, and $\Theta_2 \vdash p_2 : \hat{\theta}$, where Θ_1 and Θ_2 are subsets of Θ such that $\Theta_1 \perp \Theta_2$. By the induction hypothesis for p_1 and the definition of $\leq$ for procedural types, there are Θ_1', $\hat{\theta}'$, and θ' such that either $\langle \Theta_1', \hat{\theta}' \xrightarrow[P]{} \theta' \rangle$ or $\langle \Theta_1', \hat{\theta}' \longrightarrow \theta' \rangle$ belong to $\Psi(\Theta_0, p_1)$, $\Theta_1' \subseteq \Theta_1$, $\hat{\theta} \leq \hat{\theta}'$, and $\theta' \leq \theta$.

Since $\hat{\theta} \leq \hat{\theta}'$, for any $\theta'' \in \hat{\theta}'$ there will be a $\bar{\theta} \in \hat{\theta}$ such that $\bar{\theta} \leq \theta''$ and, since $\Theta_2 \vdash p_2 : \hat{\theta}$, $\Theta_2 \vdash p_2 : \bar{\theta}$. Then by the induction hypothesis for p_2 there is a $\langle \bar{\Theta}', \bar{\theta}' \rangle \in \Psi(\Theta_0, p_2)$ such that $\bar{\Theta}' \subseteq \Theta_2$ and $\bar{\theta}' \leq \bar{\theta}$. Let η be a function mapping each $\theta'' \in \hat{\theta}'$ into such a pair $\langle \bar{\Theta}', \bar{\theta}' \rangle$. Then η is a function from $\hat{\theta}'$ to $\Psi(\Theta_0, p_2)$ such that $[\eta\theta'']_1 \subseteq \Theta_2$ and $[\eta\theta'']_2 \leq \theta''$ hold for all $\theta'' \in \hat{\theta}'$. Let $\Theta_2' = \bigcup_{\theta'' \in \hat{\theta}'} [\eta\theta'']_1$, so that $\Theta_2' \subseteq \Theta_2$.

Since $\Theta_1' \subseteq \Theta_1$, $\Theta_2' \subseteq \Theta_2$, and $\Theta_1 \perp \Theta_2$, we have $\Theta_1' \perp \Theta_2'$ by Proposition 9. This completes the conditions needed to show that, by the definition of Ψ,

$$\langle \Theta_1' \cup \Theta_2', \theta' \rangle \in \Psi(\Theta_0, p_1p_2)$$

and also $\Theta_1' \cup \Theta_2' \subseteq \Theta$ and $\theta' \leq \theta$. (*End of Proof*)

To illustrate type checking, we consider a example that is similar to one of the problematic examples at the end of [7]. Suppose $[\, n\colon \{\textbf{int exp}\}\,] \subseteq \Theta_0$. Then

$$\langle [\, n\colon \{\textbf{int exp}\}\,], \textbf{int exp}\rangle \in \Psi(\Theta_0, n)$$

$$\langle [\, n\colon \{\textbf{int exp}\}\,], \textbf{int exp}\rangle \in \Psi(\Theta_0, n+1)$$

$$\langle [\, n\colon \{\textbf{int exp}\}\,], a\colon \textbf{int exp}\rangle \in \Psi(\Theta_0, \langle a \equiv n+1, b \equiv \cdots\rangle)$$

$$\langle [\, n\colon \{\textbf{int exp}\}\,], \textbf{int exp}\rangle \in \Psi(\Theta_0, \langle a \equiv n+1, b \equiv \cdots\rangle.a)\ ,$$

where $\cdots$ can be any phrase, even one that has no typing. Moreover, if we abstract on any identifier other than n, we get

$$\langle [\, n\colon \{\textbf{int exp}\}\,], \{\} \xrightarrow{P} \textbf{int exp}\rangle \in \Psi(\Theta_0, \lambda c\colon \cdots.\ \langle a \equiv n+1, b \equiv \cdots\rangle.a)$$

$$\langle [\, n\colon \{\textbf{int exp}\}\,], \textbf{int exp}\rangle \in \Psi(\Theta_0, (\lambda c\colon \cdots.\ \langle a \equiv n+1, b \equiv \cdots\rangle.a)(\cdots))\ .$$

Thus each term of the reduction sequence

$$(\lambda c\colon \cdots.\ \langle a \equiv n+1, b \equiv \cdots\rangle.a)(\cdots) \Longrightarrow \langle a \equiv n+1, b \equiv \cdots\rangle.a \Longrightarrow n+1$$

takes on the type **int exp** under any type assignment containing $[\, n\colon \{\textbf{int exp}\}\,]$.

6. The Remaining Problems

Beyond the progress reported here, much remains to be done:

- The efficiency of the typechecking algorithm needs to be understood and, if possible, improved.
- There is need for an alternative form of procedure that can interfere with its argument. Such a construct seems to be necessary to define active procedures recursively (as can be seen by considering the right side of the fixed-point equation $Yf = f(Yf)$ when f is active). Another motivation is the desire to regard assignment as a procedure call (as in [6]), so that $x := x + 1$ becomes an abbreviation for $x(x + 1)$. We speculate that such procedures might be obtained by abstracting on qualified identifiers (e.g. on $x.a$ rather than simply x).
- A substantial generalization is needed to deal with **goto**'s, escapes, or other operations that require continuation semantics.
- A semantic model is needed that will make it evident that distinct identifiers possess noninterfering meanings.

Despite these problems, however, the present work illustrates the utility of conjunctive types. It seems possible that their application to the syntactic control of interference may generalize to the syntactic treatment of a variety of program properties.

Acknowledgements The author wishes to thank Bob Tennent and Steve Brookes for their encouragement, and Mary and Edward Reynolds for their patience.

References

[1] Brinch Hansen, P. *Structured Multiprogramming.* **Communications of the ACM**, vol. 15 (1972), pp. 574–578.

[2] Coppo, M., Dezani-Ciancaglini, M., and Venneri, B. *Functional Characters of Solvable Terms.* **Zeitschrift für Mathematische Logik und Grundlagen der Mathematik**, vol. 27 (1981), pp. 45–58.

[3] Hoare, C. A. R. *Monitors: An Operating System Structuring Concept.* **Communications of the ACM**, vol. 17 (1974), pp. 549–557.

[4] Hoare, C. A. R. *Towards a Theory of Parallel Programming.* In: **Operating Systems Techniques**, edited by C. A. R. Hoare and R. H. Perrott. Academic Press, London, 1972, pp. 61–71.

[5] Reynolds, J. C. *The Essence of Algol.* In: **Algorithmic Languages**, edited by J. W. de Bakker and J. C. van Vliet. North-Holland, Amsterdam, 1981, pp. 345–372.

[6] Reynolds, J. C. *Preliminary Design of the Programming Language Forsythe.* Report, no. CMU-CS-88-159, Carnegie Mellon University, Computer Science Department, June 1988.

[7] Reynolds, J. C. *Syntactic Control of Interference.* In: **Conference Record of the Fifth ACM Symposium on Principles of Programming Languages**, Tucson. 1978, pp. 39–46.