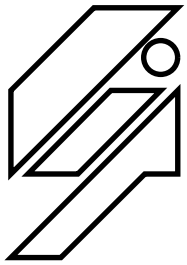
Deriving Proof Rules Form Continuations Semantics

Philippe Audebaud, Elena Zucca



Laboratoire de l'Informatique du Parallélisme

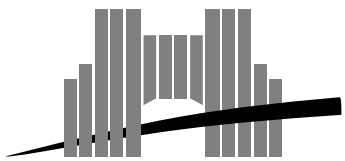
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Deriving Proof Rules from Continuation Semantics

Philippe Audebaud
Elena Zucca

June 27, 1997

Research Report N^{O} RR97-19



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) (0)4.72.72.80.00 Télécopieur : (+33) (0)4.72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Deriving Proof Rules from Continuation Semantics

Philippe Audebaud
Elena Zucca

June 27, 1997

Abstract

We claim that the continuation style semantics of a programming language can provide a starting point for constructing a proof system for that language. The basic idea is to see weakest precondition as a particular instance of continuation style semantics, hence to interpret correctness assertions (e.g. Hoare triples $\{p\}\ C\ \{r\}$) as inequalities over continuations. This approach also shows a correspondence between labels in a program and annotations.

Keywords: Continuations, Hoare Semantics, Exceptions, Labels

Résumé

Nous montrons comment la sémantique par continuations peut servir de point de départ pour construire un système de preuves pour ce langage. L'idée clé est de voir la "plus faible précondition" comme une sémantique par continuations particulière, puis d'interpréter les assertions à la Hoare ($\{p\}\ C\ \{r\}$) comme des inégalités portant sur les continuations. Cette approche permet également d'établir une correspondance entre étiquettes dans un programme et annotations.

Mots-clés: continuations, sémantique à la Hoare, exceptions, étiquettes

Deriving Proof Rules from Continuation Semantics

Philippe Audebaud
LIP - Laboratoire de l'Informatique du Parallélisme
ENS Lyon, 46 allée d'Italie, 69007 Lyon, France
email: Philippe.Audebaud@ens-lyon.fr

Elena Zucca
DISI - Dip. di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35, 16146 Genova, Italy
email: zucca@disi.unige.it

June, 1997

Abstract

We claim that the continuation style semantics of a programming language can provide a starting point for constructing a proof system for that language. The basic idea is to see weakest precondition as a particular instance of continuation style semantics, hence to interpret correctness assertions (e.g. Hoare triples $\{p\}\ C\ \{r\}$) as inequalities over continuations. This approach also shows a correspondence between labels in a program and annotations.

Introduction

In the old 70's already, Jensen [7] noted a strong resemblance between continuation style semantics and weakest precondition. To see the point, consider e.g. the semantic clause usually given for command sequence in continuation style (see e.g. [11, 10]).

$$[\![C_1; C_2]\!]_{cs}\, k = [\![C_1]\!]_{cs}([\![C_2]\!]_{cs}\, k), \text{ for any } k \text{ continuation}$$

and the corresponding clause defining weakest precondition [5]:

$$wlp(C_1; C_2)\ r = wlp(C_1)(wlp(C_2)\ r), \text{ for any } r \text{ postcondition.}$$

The two clauses are formally the same, and an analogous remark can be made for other kinds of commands.

Anyway this similarity, in [7] and later in [11], which proposes its verification as an exercise, has only been noticed as a kind of coincidence and, for what we know, never developed in the next-coming literature.

We believe instead that it would be worthwhile to regard this relation as more substantial and potentially contributing to clarify the role of both the approaches in understanding programming languages. In particular, we think that this subject deserves a renewed interest now that continuations have been recognized as an important unifying paradigm for many areas of computer science (see e.g. [1] for a survey). In this paper, our aim is to make a first step in this direction.

To this end, we first of all show that axiomatic semantics is just an instance of continuation semantics, obtained taking as "answers", in the continuation style sense, booleans with a suitable order (indeed with this choice continuations turn out to be predicates on states).

A consequence of this fact is that the continuation style semantics of a programming language can be taken as starting point for constructing a proof system for that language. More precisely, we can introduce judgments of the form $\vdash \{p\}\ C\ \{r\}$ where p, r are continuations and C is a

command, corresponding to Hoare triples in their extensional version, i.e. where one does not care about the syntactic representation of predicates [10]. These judgments are interpreted as inequalities $p \geq [\![C]\!]_{cs} r$ with $\geq$ the pointwise partial order over continuations; hence, the semantic clauses provide a guideline for constructing a proof system for such judgments.

We begin (Sect. 1) illustrating the idea on the standard imperative language for which weakest precondition is usually defined (***While*** language in the following). As one can expect, the proof system we get in this case turns out to be Hoare's system. Then (Sect. 2, Sect. 3), we consider two extensions of the language allowing to change the control flow (exceptions and goto's), for which the continuation semantics is especially well-suited. In this case, the application of our idea has some interesting consequences.

Since the continuation style semantics for these languages uses an environment associating continuations with labels, the proof systems we get have judgments of the form $\eta \vdash \{p\}\ C\ \{r\}$ where η is an environment associating predicates with labels.

These proof systems are formally different from other extensions of Hoare's logic to jumps proposed in the literature [4, 3, 9], which all introduce more complex forms of postconditions in correctness assertions; indeed, our analysis shows that these proposals are based on a direct style view of the language (which of course requires a more complex model when control flow is not always sequential).

On the contrary, the proof systems we get are a natural generalization of the standard case. For instance, the rule for a block of labelled commands `begin` $\alpha_1 : C_1; \ldots; \alpha_n : C_n$ `end` is exactly the rule for `while` B `do` C in the case in which the block is `begin` α : `if` B `then` C; `goto` α `else skip end`.

Moreover, our approach shows a correspondence between labels and annotations in programs. A label, say α, is used to denote a specific point in a program to which it is possible to jump during the execution. An annotation, say P, is a formula (i.e. a syntactic representation of a predicate) inserted in a specific point of a program to denote that some property is expected to hold there.

Now, in the continuation style semantics, if $\alpha : C$ is a labelled command in a program, then the denotation of α in the current environment η is $[\![C]\!]_{cs}\, \eta\, r$, with r the continuation of C; in the particular case where continuations are predicates, $[\![C]\!]_{cs}\, \eta\, r$ is the weakest precondition of C w.r.t. the postcondition r (under η), hence we can correctly insert before C an annotation representing $\eta(\alpha)$ (or a stronger condition).

Then, the problem of finding, for a given program, a set of annotations sufficient to prove its correctness can be reduced to the problem of finding a tuple of predicates which is a pre-fixpoint of the functional which defines the continuation style semantics of the program and which can be expressed within the given language of formulas.

In Sect. 4, we illustrate this idea by giving an algorithm which, inputs a correctness assertion $\{P\}\ C\ \{R\}$ (in the intensional version, i.e. with P, R formulas), and outputs a formula Q_0 and a set of constraints $\{X_i \Rightarrow Q_i \mid i \in 1..n\}$, with $Q_0, Q_1, \ldots, Q_n$ formulas with free variables (indeterminates) $X_1, \ldots, X_n$. The algorithm output is such that each solution (namely, ground substitution $\sigma = \{X_i \mapsto S_i \mid i \in 1..n\}$ s.t. $S_i \Rightarrow Q_i\langle\sigma\rangle$, $i \in 1..n$) gives a set of correct annotations for C w.r.t. R; in particular $Q_0\langle\sigma\rangle$ is a correct precondition for C w.r.t. to R, hence $\{P\}\ C\ \{R\}$ is valid if $P \Rightarrow Q_0\langle\sigma\rangle$.

Finally, the conclusion provides a more detailed comparison with related works and outlines some further research directions.

1 Deriving Hoare's System

In this section, we illustrate our ideas on the ***While*** language handled in the original Hoare's system.

1.1 Continuation Semantics of *While*

First of all, we recall the well-known continuation style semantics of the ***While*** language. We report direct semantics too for completeness. Here below our chosen notations.

Notations. We write $(A \to B)$ for the set of the functions from A into B. Application of f to x is denoted by $f\ x$, and $\circ$ denotes function composition. For any set A, id_A denotes the identity of A. If A and B are two cpo's with bottom element $\bot_A$ and $\bot_B$, respectively, then we denote by $[A \to B]$ the cpo of the continuous functions from A into B with the point-wise partial order, and by $[A \to_\bot B]$ the cpo of the continuous strict functions (i.e. f such that $f\ \bot_A = \bot_B$). Finally, for $\Phi \in [A \to A]$, $fix(\Phi)$ denotes the least fixpoint of Φ (recall that $fix(\Phi) = \vee_{n=0}^{\infty} \Phi^n \bot_A$).

The direct and continuation style semantics of the ***While*** is reported on Fig. 1. We denote by *Com*, *Expr*, *BExpr*, *Id* the sets of commands, expressions, boolean expressions and identifiers of the language, ranged over by C, E, B and I, respectively; by *Val* the set of values of expressions, and by *Answer* a cpo with bottom element $\bot_{Answer}$. Only syntax and semantics of commands are specified.

We recall the standard results about the well-definedness of continuation style semantics of ***While*** and its relation with direct style semantics (see e.g. [11, 10] for the proof).

Proposition 1.1 *For any command C of* ***While****,*

1. $[\![C]\!]_{cs} \in [Cont \to Cont]$;
2. *for any $r \in Cont$, $[\![C]\!]_{cs} r = r \circ [\![C]\!]_{ds}$.*

1.2 Predicates as Continuations

Take as *Answer* the cpo *Bool* of booleans values $true, false$ with the (unusual) order $\leq$ induced by $true < false$. This way, $\bot_{Answer} = true$. This amounts to interpreting $p \geq q$ as the boolean implication $p \Rightarrow q$. We set $Pred = [State_\bot \to_\bot Bool]$; continuations can be seen in this case as *predicates*, and the resulting point-wise partial order in *Pred* is that $p \geq q$ iff p is a stronger condition than q.

Let us now recall the definition of weakest (liberal) precondition *wlp* of a command C w.r.t. a postcondition r (where $r \in (State \to \{true, false\})$)

$$wlp(C)r\ s = true \quad \text{iff} \quad [\![C]\!]_{ds}\, s = \bot \text{ or } [\![C]\!]_{ds}\, s = s', s' \neq \bot \text{ and } r\ s' = true$$

An equivalent more concise definition can be given taking $r \in [State_\bot \to_\bot Bool]$,

$$wlp(C)r\ s = r\ ([\![C]\!]_{ds} s).$$

Note that the assumption that continuations (predicates in this case) are strict corresponds to the intuitive idea that a non terminating command verifies any postcondition, since we are considering partial correctness. Now, from Prop. 1.1, we easily get that:

Fact 1.2 *For any command C, $[\![C]\!]_{cs} = wlp(C)$.*

Indeed, it is easy to see that, with this choice of *Answer*, semantic clauses in continuation style are exactly usual clauses defining weakest precondition.

Boolean connectives. Usual boolean connectives can be easily extended to *Pred*, as summarized below.

- $True\, s = true$, for any $s \in State_\bot$;

Syntax

$C ::= \texttt{skip} \mid I := E \mid C_1 ; C_2 \mid \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \mid \texttt{while } B \texttt{ do } C$

States

$State = (Id \to Val)$

$State_\perp = State \cup \{\perp_{State}\}$

$- [-/-] : State_\perp \times Val \times Id \to State_\perp$ substitution on states

$\perp_{State} [v/I] = \perp_{State}$,

$s [v/I]\ I = v$, for $s \neq \perp_{State}$,

$s [v/I]\ I' = s\ I'$, for $s \neq \perp_{State}$, $I \neq I'$

Direct style semantics

$[\![-]\!]_{ds} : Com \to [State_\perp \to_\perp State\ with Bottom]$

$[\![\texttt{skip}]\!]_{ds} = id_{State_\perp}$

$[\![I := E]\!]_{ds} = \lambda s.s\,[[\![E]\!]\,s/I]$

$[\![C_1 ; C_2]\!]_{ds} = [\![C_2]\!]_{ds} \circ [\![C_1]\!]_{ds}$

$[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!]_{ds} = cond_{State_\perp}([\![B]\!], [\![C_1]\!]_{ds}, [\![C_2]\!]_{ds})$

$[\![\texttt{while } B \texttt{ do } C]\!]_{ds} = fix(\lambda f.\ cond_{State_\perp}([\![B]\!], f \circ [\![C]\!]_{ds}, id_{[State_\perp \to_\perp State_\perp]}))$

Continuation semantics

$Cont = [State_\perp \to_\perp Answer]$

$[\![-]\!]_{cs} : Com \to [Cont \to Cont]$

$[\![\texttt{skip}]\!]_{cs} = id_{Cont}$

$[\![I := E]\!]_{cs} = \lambda r.r\,[[\![E]\!]/I]$

$[\![C_1 ; C_2]\!]_{cs} = [\![C_1]\!]_{cs} \circ [\![C_2]\!]_{cs}$

$[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!]_{cs} = \lambda r.\ cond_{Answer}([\![B]\!], [\![C_1]\!]_{cs}\ r, [\![C_2]\!]_{cs}\ r)$

$[\![\texttt{while } B \texttt{ do } C]\!]_{cs} = fix(\lambda f.\lambda r.\ cond_{Answer}([\![B]\!], [\![C]\!]_{cs}(f\ r), r))$

Auxiliary functions

$- [-/-] : Cont \times (State \to Val) \times Id \to Cont$ substitution on continuations

$q [e/I] = \lambda s.q\ s\ [e(s)/I]$

$cond_{Answer} : [State_\perp \to_\perp Bool] \times [State_\perp \to_\perp Answer] \times [State_\perp \to_\perp Answer] \to [State_\perp \to_\perp Answer]$

(we omit the suffix when clear from the context)

$cond(b, q_1, q_2)\ \perp_{State} = \perp_{Answer}$,

$cond(b, q_1, q_2)\ s = q_1\ s$, if $s \neq \perp_{State}$ and $b\ s = true$

$cond(b, q_1, q_2)\ s = q_2\ s$, if $s \neq \perp_{State}$ and $b\ s = false$

Figure 1: Direct and continuation style semantics for the ***While*** language.

- $False \perp = true, False\, s = false$, for any $s \neq \perp$;
- For any $q \in Pred$, $(\neg q)\perp = true, (\neg q)s = \neg(q\ s)$, for any $s \neq \perp$;
- For any $q_1, q_2 \in Pred$, $(q_1 \wedge q_2)\perp = true, (q_1 \wedge q_2)s = q_1 s \wedge q_2 s$, for any $s \neq \perp$; $\vee$ and $\Rightarrow$ can be defined analogously ($q_1 \Rightarrow q_2$ can be equivalently defined as $(\neg q_1) \vee q_2$ as usual).

Note that *True* and *False* are the bottom and top element of *Pred*, respectively. Moreover, we have $q_1 \Rightarrow q_2$ iff $q_1 \geq q_2$. Finally, it is easy to see that the following fact holds.

Fact 1.3 *1.* $cond_{Bool}(b, q_1, q_2) = (b \wedge q_1) \vee (\neg b \wedge q_2)$.

2. $p \Rightarrow cond_{Bool}(b, q_1, q_2)$ *iff* $p \wedge b \Rightarrow q_1$ *and* $p \wedge \neg b \Rightarrow q_2$.

1.3 Hoare's System

We define now *correctness assertions* as triples $\langle p, C, r\rangle$, written $\{p\}\ C\ \{r\}$, where p, r are continuations and C is a command. We say that a correctness assertion $\{p\}\ C\ \{r\}$ is *valid*, and write $\models \{p\}\ C\ \{r\}$, iff $p \geq [\![C]\!]_{cs} r$. Our subsequent aim is to derive, starting from the semantic clauses, a (sound and complete w.r.t. validity) proof system having judgments of the form $\vdash \{p\}\ C\ \{r\}$. To this end, we find an equivalent condition for $p \geq [\![C]\!]_{cs} r$, for each kind of command.

Proposition 1.4 *For any pair of continuations* p, r

1. $p \geq [\![\texttt{skip}]\!]_{cs} r$ *iff* $p \geq r$;

2. $p \geq [\![I \texttt{ := } E]\!]_{cs} r$ *iff* $p \geq r\,[[\![E]\!]/I]$;

3. $p \geq [\![C_1 \texttt{;}\ C_2]\!]_{cs} r$ *iff* $p \geq [\![C_1]\!]_{cs} q$ *and* $q \geq [\![C_2]\!]_{cs} r$, *for some* $q \in Cont$;

4. $p \geq [\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!]_{cs} r$ *iff* $p \wedge [\![B]\!] \geq [\![C_1]\!]_{cs} r$ *and* $p \wedge \neg[\![B]\!] \geq [\![C_2]\!]_{cs} r$;

5. $p \geq [\![\texttt{while } B \texttt{ do } C]\!]_{cs} r$ *iff* $p \geq i$, $i \wedge [\![B]\!] \geq [\![C]\!]_{cs} i$ *and* $i \wedge \neg[\![B]\!] \geq r$, *for some* $i \in Cont$.

Proof.

1. Immediate.
2. Immediate.
3. From the semantic clause, $p \geq [\![C_1 \texttt{;}\ C_2]\!]_{cs} r$ implies $p \geq [\![C_1]\!]_{cs} q$ and $q \geq [\![C_2]\!]_{cs} r$, for some $q \in Cont$ ($q = [\![C_2]\!]_{cs}\, r$). Anyway, it is easy to see that the opposite implication also holds, since $[\![C_1]\!]_{cs}$ is monotonic (Prop. 1.1-(1)). Hence the result.
4. From the semantic clause,

$$\begin{array}{ll} & p \geq [\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!]_{cs} r \\ \text{iff} & p \geq cond([\![B]\!], [\![C_1]\!]_{cs}\, r, [\![C_2]\!]_{cs}\, r) \\ \text{iff} & p \wedge [\![B]\!] \geq [\![C_1]\!]_{cs} r \text{ and } p \wedge \neg[\![B]\!] \geq [\![C_2]\!]_{cs} r \end{array}$$

   The result follows.
5. Set $w = [\![\texttt{while } B \texttt{ do } C]\!]_{cs}$. From the semantic clause, w is the least fixpoint of the functional Φ defined by

$$\Phi\ f\ r = cond([\![B]\!], [\![C]\!]_{cs}(f\ r), r).$$

   Whence, $w\ r$ is the least fixpoint of the continuous function ϕ defined by

$$\phi\ q = cond([\![B]\!], [\![C]\!]_{cs} q, r)$$

(Skip)	$\dfrac{p \Rightarrow r}{\{p\}\ \texttt{skip}\ \{r\}}$
(Assign)	$\dfrac{p \Rightarrow r\,[[\![E]\!]/I]}{\{p\}\ I\ \texttt{:=}\ E\ \{r\}}$
(Conc)	$\dfrac{\{p\}\ C_1\ \{q\} \quad \{q\}\ C_2\ \{r\}}{\{p\}\ C_1\,;\ C_2\ \{r\}}$
(If)	$\dfrac{\{p \wedge [\![B]\!]\}\ C_1\ \{r\} \quad \{p \wedge \neg [\![B]\!]\}\ C_2\ \{r\}}{\{p\}\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \{r\}}$
(While)	$\dfrac{p \Rightarrow i \quad \{i \wedge [\![B]\!]\}\ C\ \{i\} \quad i \wedge \neg [\![B]\!] \Rightarrow r}{\{p\}\ \texttt{while}\ B\ \texttt{do}\ C\,\{r\}}$

Table 1: Proof system for the ***While*** language.

Set $\bar{\imath} = w\ r$, and $b = [\![B]\!]$. Since $\bar{\imath}$ is a fixpoint, $\bar{\imath} = \mathit{cond}(b, [\![C]\!]_{cs}\,\bar{\imath}, r)$, which implies $\bar{\imath} \wedge b \geq [\![C]\!]_{cs}\bar{\imath}$ and $\bar{\imath} \wedge \neg b \geq r$. Hence the necessary condition.
It is easy to see that the opposite implication also holds. Indeed, if $i \wedge b \geq [\![C]\!]_{cs} i$ and $i \wedge \neg b \geq r$ hold for some i, then $i \geq \mathit{cond}(b, [\![C]\!]_{cs}\, i, r) = \phi\ i$, which means that i is a pre-fixpoint of the function ϕ. Then, since $\bar{\imath} = w\ r$ is the least fixpoint of ϕ, $i \geq w\ r$, hence by transitivity $p \geq w\ r$, hence the result. □

Altogether, we get the following result.

Proposition 1.5 *If* $\vdash \{-\} - \{-\} \subseteq \mathit{Cont} \times \mathit{Com} \times \mathit{Cont}$ *is the relation inductively defined by the set of rules in Table 1, then, for any command* C *of the* ***While*** *language,* p, r *continuations,*

$$\vdash \{p\}\ C\ \{r\} \quad \textit{iff} \quad \models \{p\}\ C\ \{r\}$$

Proof. By structural induction and Prop. 1.4. □

The proof system in Fig. 1 is Hoare's system in its extensional version , i.e. where preconditions and postconditions in Hoare's triples are assumed to be predicates (semantic entities) and not formulas (syntactic representations of predicates), hence a result of completeness can be obtained (see e.g. [10] for a discussion about this difference). Hence, Prop. 1.4 states nothing new. Anyway, note that the results and the proof itself are formulated using, as long as possible, only general properties of the continuation style semantics; the only point where the particular choice of predicates as continuations plays a role is in allowing to rewrite the conditional operator in terms of the logical connectives.

Analogously, the following consequence rule

$$\text{(Conseq)} \qquad \frac{p' \Rightarrow p \quad \{p\}\ C\ \{r\} \quad r \Rightarrow r'}{\{p'\}\ C\ \{r'\}},$$

which is given in some equivalent formulation of Hoares's system, can be easily obtained from the general properties of the continuation style semantics, as shown below.

Fact 1.6 *For any p, p', r, r' continuations s.t. $p' \geq p$ and $r \geq r'$, $\{p\}\ C\ \{r\}$ implies $\{p'\}\ C\ \{r'\}$ for any command C.*

Proof. The fact that $p' \geq [\![C]\!]_{cs}\, r$ holds by transitivity; the fact that $[\![C]\!]_{cs}\, r \geq [\![C]\!]_{cs}\, r'$ holds since $[\![C]\!]_{cs}$ is monotonic. □

2 Proof Rules for Exceptions

In this section and the following one, we extend the language ***While*** by constructs allowing a non sequential control of flow and show that, again, the continuation style semantics leads to a proof system.

Let us first consider an exception mechanism through an extension ***Excp*** of the ***While*** language with two commands: `trap` and `raise`.

Informally, the execution of $\texttt{trap}\,\alpha\,\texttt{in}\,C_1\,\texttt{with}\,C_2$ consists in the execution of C_1 until some $\texttt{raise}\,\alpha$ command is encountered; in this case, the normal continuation is abandoned and the command C_2 (called a *handler*) is executed instead. We call α a *label* for keeping a uniform terminology w.r.t. the goto's case illustrated in the next section.

The continuation style semantics of ***Excp*** is given in Fig. 2. We denote by *Label* the set of labels (exception names in this case), ranged over by α, and by *ContEnv* the set of the environments associating continuations with labels, ranged over by η.

The view of continuation style semantics as axiomatic semantics still holds in the following sense. The notion of weakest precondition of a command w.r.t. a postcondition is now parameterized by an environment associating predicates with labels: $[\![C]\!]_{cs}\, \eta\, r$ is the weakest precondition of C w.r.t. r under η. Indeed, whenever C contains a $\texttt{raise}\,\alpha$ command with α free, the weakest precondition of C depends on the weakest precondition w.r.t. r of the corresponding handler, which is given by η.

For η, η' environments, set $\eta \leq \eta'$ iff $\forall x\ \eta(x) \leq \eta'(x)$. The following proposition states that the continuation style semantics of ***Excp*** is well-defined and associates with each command a function monotonic in its first argument (the environment).

Proposition 2.1 *For any command C of **Excp***

1. *for any environment η, $[\![C]\!]_{cs}\, \eta \in [Cont \to Cont]$;*
2. *if $\eta \leq \eta'$ then $[\![C]\!]_{cs}\, \eta \leq [\![C]\!]_{cs}\, \eta'$.*

Proof. By structural induction, using the fact that the function $\eta \mapsto [\![C]\!]_{cs}\, \eta$ is continuous. □

Analogously to what we have done for the ***While*** language, we now derive from the semantic clauses a proof system for the extended language. In this case, according to the continuation style semantics of a command, the validity of correctness assertions depends on a continuation environment: we say that $\{p\}\ C\ \{r\}$ is *valid under* η, and write $\eta \models \{p\}\ C\ \{q\}$, iff $p \geq [\![C]\!]_{cs}\eta\ r$.

Analogously, judgments of the proof system are of the form $\eta \vdash \{p\}\ C\ \{r\}$.

Proposition 2.2 *For any pair of continuations p, r and any continuation environment η:*

1. *$p \geq [\![\texttt{raise}\,\alpha]\!]_{cs}\eta\ r$ iff $p \geq \eta(\alpha)$;*
2. *$p \geq [\![\texttt{trap}\,\alpha\,\texttt{in}\,C_1\,\texttt{with}\,C_2]\!]_{cs}\eta\ r$ iff $q \geq [\![C_2]\!]_{cs}\eta\ r$ and $p \geq [\![C_1]\!]_{cs}\eta[q/\alpha]\ r$, for some $q \in Cont$.*

Proof. For each case:

Syntax

$C ::= \ldots \mid \texttt{raise}\,\alpha \mid \texttt{trap}\,\alpha\,\texttt{in}\,C_1\,\texttt{with}\,C_2$

Continuation environments

$ContEnv = (Label \to Cont)$

$-[-/-] : ContEnv \times Cont \times Label \to ContEnv$ substitution on continuation environments

$\eta[q/\alpha](\alpha) = q,$

$\eta[q/\alpha](\beta) = \eta(\beta)$, for $\beta \neq \alpha$

$\eta[q_1/\alpha_1, \ldots, q_n/\alpha_n]$ stands for $\eta[q_1/\alpha_1]\ldots[q_n/\alpha_n]$

Continuation semantics

$[\![-]\!]_{cs} : Com \to (ContEnv \to [Cont \to Cont])$

$[\![\texttt{skip}]\!]_{cs}\,\eta = id_{Cont}$

$[\![I := E]\!]_{cs}\,\eta = \lambda r.r\,[[\![E]\!]/I]$

$[\![C_1; C_2]\!]_{cs}\,\eta = [\![C_1]\!]_{cs}\,\eta \circ [\![C_2]\!]_{cs}\,\eta$

$[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!]_{cs}\,\eta = cond([\![B]\!]_{ds}, [\![C_1]\!]_{cs}\,\eta, [\![C_2]\!]_{cs}\,\eta)$

$[\![\texttt{while}\ B\ \texttt{do}\ C]\!]_{cs}\,\eta = fix(\lambda f.\lambda r.\ cond([\![B]\!]_{cs}, [\![C]\!]_{cs}\,\eta(f\ r), r))$

$[\![\texttt{raise}\,\alpha]\!]_{cs}\,\eta = \lambda r.\eta(\alpha)$

$[\![\texttt{trap}\ \alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2]\!]_{cs}\,\eta = \lambda r.\,[\![C_1]\!]_{cs}\eta\,[([\![C_2]\!]_{cs}\,\eta\ r)/\alpha]\ \ r$

Figure 2: Continuation style semantics of the ***Excp*** language

(Raise)	$\dfrac{p \Rightarrow \eta(\alpha)}{\eta \vdash \{p\}\ \texttt{raise}\,\alpha\ \{r\}}$
(Trap)	$\dfrac{\eta[q/\alpha] \vdash \{p\}\ C_1\ \{r\} \quad \eta \vdash \{q\}\ C_2\ \{r\}}{\eta \vdash \{p\}\ \texttt{trap}\ \alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2\ \{r\}}$

Table 2: Additional rules for the ***Excp*** language

1. Immediate.
2. From the semantic clause, $p \geq [\![\texttt{trap}\ \alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2]\!]_{cs}\eta\ r$ implies $p \geq [\![C_1]\!]_{cs}(\eta[q/\alpha])\ r$, for some q ($q = [\![C_2]\!]_{cs}\,\eta\ r$). Anyway, it is easy to see that the opposite implication also holds. Indeed, set $\bar{q} = [\![C_2]\!]_{cs}\,\eta\ r$; if $p \geq [\![C_1]\!]_{cs}(\eta[q/\alpha])\ r$, for some q s.t. $q \geq \bar{q}$, then $\eta[q/\alpha] \geq \eta[\bar{q}/\alpha]$ since the order on $ContEnv$ is defined pointwise; then $[\![C_1]\!]_{cs}\,\eta[q/\alpha] \geq [\![C_1]\!]_{cs}\,\eta[\bar{q}/\alpha]$ since $[\![C_1]\!]_{cs}$ is monotonic (Lemma 2.1-(2)), and in particular $[\![C_1]\!]_{cs}\,\eta[q/\alpha]\,r \geq [\![C_1]\!]_{cs}\,\eta[\bar{q}/\alpha]\,r$. Hence the result. □

From the above proposition, it is immediate to derive the proof rules for `raise` and `trap` commands, given in Table 2. Note that by the rule (Raise) we can deduce in particular the judgment $\eta \vdash \{\eta(\alpha)\}\ \texttt{raise}\,\alpha\ \{False\}$, expressing the fact that after a `raise` command the normal continuation (postcondition) is abandoned.

For the commands of the ***While*** language, the continuation environment is not significant and must be simply propagated from the premises to the consequence of proof rules. Formally, let us keep the same names for the new proof rules obtained by replacing every judgment of the form $\vdash \{p\}\ C\ \{r\}$ by a judgment $\eta \vdash \{p\}\ C\ \{r\}$. We can state a soundness and completeness result for

Syntax

$C ::= \ldots \mid \texttt{goto}\,\alpha \mid \texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n\,\texttt{end}$

Continuation semantics

$[\![\texttt{goto}\,\alpha]\!]_{cs}\,\eta\,r = \eta(\alpha)$

$[\![\texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n\,\texttt{end}]\!]_{cs}\,\eta\,r = \bar{q}_1$, where $\langle \bar{q}_1, \ldots, \bar{q}_n \rangle = \mathit{fix}(\Phi)$,

$\Phi\langle q_1, \ldots, q_n \rangle \equiv \langle [\![C_1]\!]_{cs}\,\bar{\eta}\,q_2, \ldots, [\![C_n]\!]_{cs}\,\bar{\eta}\,r \rangle$, with $\bar{\eta} \equiv \eta[q_1/\alpha_1, \ldots, q_n/\alpha_n]$.

Figure 3: Continuation style semantics of the ***Goto*** language

(Goto) $$\frac{p \Rightarrow \eta(\alpha)}{\eta \vdash \{p\}\,\texttt{goto}\,\alpha\,\{r\}}$$

(Block) $$\frac{p \Rightarrow q_1 \quad \bar{\eta} \vdash \{q_1\}\,C_1\,\{q_2\} \quad \cdots \quad \bar{\eta} \vdash \{q_n\}\,C_n\,\{r\}}{\eta \vdash \{p\}\,\texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n\,\texttt{end}\,\{r\}} \quad \bar{\eta} \equiv \eta[q_1/\alpha_1, \ldots, q_n/\alpha_n]$$

Table 3: Additional rules for the ***Goto*** language

the extended proof system.

Proposition 2.3 *If $- \vdash \{-\} - \{-\} \subseteq ContEnv \times Cont \times Com \times Cont$ is the relation inductively defined by the previous proof rules plus (Raise) and (Trap) in Table 2, then, for any $\eta \in ContEnv$, C command of **Excp**, p, r continuations,*

$$\eta \vdash \{p\}\,C\,\{r\} \quad \textit{iff} \quad \eta \models \{p\}\,C\,\{r\}$$

Proof. By structural induction and Prop. 2.2. □

3 Proof Rules for Goto's

We consider now a language ***Goto*** which extends ***Excp*** by an unrestricted jump mechanism. Informally, the execution of a block of labelled commands $\texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n\,\texttt{end}$ consists of the execution of the sequence of commands $C_1; \ldots; C_n$ until some $\texttt{goto}\,\alpha_i$ command is encountered; in this case, the normal continuation is abandoned and the execution jumps to C_i. Note that any C_i can contain in turn blocks of labelled commands, with the usual scoping rules for labels.

The continuation style semantics of ***Goto*** is given in Fig. 3. The view of continuation style semantics as axiomatic semantics holds as already explained for ***Excp***, i.e. the weakest precondition w.r.t. r under η of a command containing some $\texttt{goto}\,\alpha$ with α free depends on $\eta(\alpha)$, which is the weakest precondition of the command labelled α w.r.t. r under η. Anyway, in this case the dependency is mutually recursive, hence the weakest precondition is defined as the least fixpoint of an equation, as for the `while` command. Correspondingly, we get the rule (Block) below, where $q_1, \ldots, q_n$ play the same role of "indeterminates" as the invariant in the rule (While).

The following proposition states that the continuation semantics of the ***Goto*** language is well-defined and associates a function monotonic in its first argument (the continuation environment), with each command.

Proposition 3.1 *For any command C of the **Goto** language*

1. *for any continuation environment* η, $[\![C]\!]_{cs}\,\eta \in [Cont \to Cont]$;

2. *if* $\eta \leq \eta'$ *then* $[\![C]\!]_{cs}\,\eta \leq [\![C]\!]_{cs}\,\eta'$.

Proof. By structural induction, using the fact that, for all $\alpha_1, \ldots, \alpha_n$ labels and η continuation environment, the function $\langle q_1, \ldots, q_n\rangle \mapsto [\![C]\!]_{cs}\,\eta\,[q_1/\alpha_1, \ldots, q_n/\alpha_n]$ is continuous. □

Again, we shall derive from the semantic clauses a proof system for the extended language, by the following proposition:

Proposition 3.2 *For any pair of continuations* p, r *and any continuation environment* η*:*

1. $p \geq [\![\texttt{goto}\,\alpha]\!]_{cs}\eta\, r$ *iff* $p \geq \eta(\alpha)$;

2. $p \geq [\![\texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n\,\texttt{end}]\!]_{cs}\eta\ r$ *iff* $p \geq q_1$, $q_1 \geq [\![C_1]\!]_{cs}\bar{\eta}\ q_2$, $\ldots$, $q_n \geq [\![C_n]\!]_{cs}\bar{\eta}\ r$, *with* $\bar{\eta} \equiv \eta[q_1/\alpha_1, \ldots, q_n/\alpha_n]$, *for some* $q_1, \ldots, q_n \in Cont$.

Proof. For each case:

1. Immediate.
2. Set$\langle \bar{q}_1, \ldots, \bar{q}_n\rangle = \mathit{fix}(\Phi)$. Since $\langle \bar{q}_1, \ldots, \bar{q}_n\rangle$ is a fixpoint of Φ (in fact, the least one),

$$\bar{q}_1 \geq [\![C_1]\!]_{cs}\,\bar{\eta}\,q_2, \quad \ldots, \quad \bar{q}_n \geq [\![C_n]\!]_{cs}\,\bar{\eta}\,r$$

with $\bar{\eta} \equiv \eta[\bar{q}_1/\alpha_1, \ldots, \bar{q}_n/\alpha_n]$. Hence, from the semantic clause, the necessary condition holds. It is easy to see that the opposite implication also holds. Indeed, assume $\langle q_1, \ldots, q_n\rangle$ is s.t. $q_i \geq [\![C_i]\!]_{cs}\,\bar{\eta}\,q_{i+1}$, for $i \in 1..n$ ($q_{n+1} = r$), where $\bar{\eta} \equiv \eta\,[q_1/\alpha_1, \ldots, q_n/\alpha_n]$. Then $\Phi_{\eta,r}\langle q_1, \ldots, q_n\rangle = \langle q_1, \ldots, q_n\rangle$; hence $\langle q_1, \ldots, q_n\rangle$ is a pre-fixpoint of Φ. Since the least fixpoint of Φ is $\langle \bar{q}_1, \ldots, \bar{q}_n\rangle$, $\langle q_1, \ldots, q_n\rangle \geq \langle \bar{q}_1, \ldots, \bar{q}_n\rangle$, hence by transitivity $p \geq \bar{q}_1$, hence the result. □

From the above proposition, it is immediate to derive the proof rules for `goto` and `block` commands, given in Table 3. Note that, as for rule (Raise), we can deduce in particular the judgment $\eta \vdash \{\eta(\alpha)\}\ \texttt{goto}\,\alpha\ \{\mathit{False}\}$, by the rule (Goto).

We can state a soundness and completeness result for the extended proof system.

Proposition 3.3 *Let* $- \vdash \{-\} - \{-\} \subseteq \mathit{ContEnv} \times \mathit{Cont} \times \mathit{Com} \times \mathit{Cont}$ *the relation inductively defined by the previous rules plus rules (Goto) and (Block) given in Table 3. Then, for any* $\eta \in \mathit{ContEnv}$, C *command of* ***Goto***, p, r *continuations,*

$$\eta \vdash \{p\}\,C\,\{r\} \quad \textit{iff} \quad \eta \models \{p\}\,C\,\{r\}.$$

Proof. By structural induction and Prop. 3.2. □

3.1 (While) and (Trap) Rules as Derived Rules

In this subsection, we show that the proof rules given for the while and trap command can be seen as a specialization of the rule given for the block of labelled commands.

To this end, we define in Table 4 a translation $\mathcal{T}$ from the whole set of the commands of ***Goto*** to the set of the commands not containing `while` and `trap` commands. We denote by $FL(C)$ the set of the free labels in a command C, defined in the usual way, and by $C\,[\beta/\gamma]$ the command obtained from C by substituting γ with β in the usual way (α conversion).

$$
\begin{array}{rcl}
\mathcal{T}(\,\texttt{skip}\,) & = & \texttt{skip} \\
\mathcal{T}(\,I\ \texttt{:=}\ E\,) & = & I\ \texttt{:=}\ E \\
\mathcal{T}(C_1\,;\,C_2) & = & \mathcal{T}(C_1)\,;\mathcal{T}(C_2) \\
\mathcal{T}(\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2) & = & \texttt{if}\ B\ \texttt{then}\ \mathcal{T}(C_1)\ \texttt{else}\ \mathcal{T}(C_2) \\
\mathcal{T}(\texttt{while}\ B\ \texttt{do}\ C) & = & \texttt{begin}\,\alpha : \texttt{if}\ B\ \texttt{then}\ \mathcal{T}(C);\texttt{goto}\,\alpha\ \texttt{else skip end} \\
 & & \text{for some } \alpha \notin FL(\mathcal{T}(C)) \\
\mathcal{T}(\texttt{raise}\,\alpha) & = & \texttt{goto}\,\alpha \\
\mathcal{T}(\texttt{trap}\,\alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2) & = & \texttt{begin}\,\beta : (\mathcal{T}(C_1\,[\gamma/\alpha]);\texttt{goto}\,\omega);\gamma : \mathcal{T}(C_2);\omega : \texttt{skip end} \\
 & & \text{for some } \beta,\gamma,\omega \notin FL(\mathcal{T}(C_1)) \cup FL(\mathcal{T}(C_2)) \\
\mathcal{T}(\texttt{goto}\,\alpha) & = & \texttt{goto}\,\alpha \\
\mathcal{T}(\texttt{begin}\,\alpha_1 : C_1;\ldots;\alpha_n : C_n\,\texttt{end}) & = & \texttt{begin}\,\alpha_1 : \mathcal{T}(C_1);\ldots;\alpha_n : \mathcal{T}(C_n)\,\texttt{end}
\end{array}
$$

Table 4: Elimination of `while` and `trap` commands

Theorem 3.4 *For any η continuation environment, C command of **Goto**, p, r continuations*

$$\eta \vdash \{p\}\ C\ \{r\} \quad \textit{iff} \quad \eta \vdash \{p\}\ \mathcal{T}(C)\ \{r\}.$$

This result could be proved in an indirect way by showing that the translation preserves the continuation style semantics of commands and by the soundness and completeness of the proof system. We give instead a direct proof which shows that the proof rules (While) and (Trap) can be obtained as derived rules. We need the following lemma.

Lemma 3.5 *For any C command of **Goto**,*

- $FL(C) = FL(\mathcal{T}(C))$;
- *for any η continuation environment, p, r continuations, $\beta \notin FL(C)$,*
  - $\eta\,[q/\beta] \vdash \{p\}\ C\ \{r\}$ *iff* $\eta \vdash \{p\}\ C\ \{r\}$;
  - $\eta\,[q/\beta] \vdash \{p\}\ C\,[\beta/\alpha]\ \{r\}$ *iff* $\eta\,[q/\alpha] \vdash \{p\}\ C\ \{r\}$.

Proof. By structural induction. □

Proof of Theorem 3.4 By structural induction. We show the non trivial cases.

While Let us consider the judgment $\eta \vdash \{p\}\ \texttt{while}\ B\ \texttt{do}\ C\{r\}$. By the translation, this judgment becomes $\eta \vdash \{p\}\ \texttt{begin}\,\alpha : \texttt{if}\ B\ \texttt{then}\ \mathcal{T}(C);\texttt{goto}\,\alpha\ \texttt{else skip end}\ \{r\}$, for some $\alpha \notin FL(C)$.

We get the following proof tree

$$\dfrac{p \Rightarrow q_1 \qquad \dfrac{\Pi_1 \quad \dfrac{q_1 \wedge [\![B]\!] \Rightarrow r}{\eta\,[q_1/\alpha] \vdash \{q_1 \wedge \neg[\![B]\!]\}\ \texttt{skip}\ \{r\}}}{\eta\,[q_1/\alpha] \vdash \{q_1\}\ \texttt{if}\ B\ \texttt{then}\ \mathcal{T}(C);\texttt{goto}\,\alpha\ \texttt{else skip}\ \{r\}}}{\eta \vdash \{p\}\ \texttt{begin}\,\alpha : \texttt{if}\ B\ \texttt{then}\ \mathcal{T}(C);\texttt{goto}\,\alpha\ \texttt{else skip end}\ \{r\}}$$

where the proof tree Π_1 is

$$\dfrac{\dfrac{\vdots}{\eta\,[q_1/\alpha] \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C)\ \{q\}} \quad \dfrac{q \Rightarrow q_1}{\eta\,[q_1/\alpha] \vdash \{q\}\ \texttt{goto}\,\alpha\ \{r\}}}{\eta\,[q_1/\alpha] \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C);\ \texttt{goto}\,\alpha\ \{r\}}$$

By Lemma 3.5 $\eta\,[q_1/\alpha] \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C)\ \{q\}$ iff $\eta \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C)\ \{q\}$. Now, it is easy to see that $\eta \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C)\ \{q\}$ and $q \Rightarrow q_1$ for some $q \in \mathit{Cont}$ iff $\eta \vdash \{q_1 \wedge [\![B]\!]\}\ \mathcal{T}(C)\ \{q_1\}$. By inductive hypothesis, from this last judgment, we get a proof for $\eta \vdash \{q_1 \wedge [\![B]\!]\}\ C\ \{q_1\}$.

In summary, we get the following proof tree:

$$\dfrac{p \Rightarrow q_1 \quad \dfrac{\vdots}{\eta \vdash \{q_1 \wedge [\![B]\!]\}\ C\ \{q_1\}} \quad q_1 \wedge \neg[\![B]\!] \Rightarrow r}{\eta \vdash \{p\}\ \texttt{while}\ B\ \texttt{do}\ C\{r\}}$$

which is the same obtained by rule (While).

Trap Let us consider the judgment $\eta \vdash \{p\}\ \texttt{trap}\ \alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2\ \{r\}$. By the translation, this judgment becomes $\eta \vdash \{p\}\ \texttt{begin}\,\beta : (\mathcal{T}(C_1\,[\gamma/\alpha]); \texttt{goto}\,\omega); \gamma : \mathcal{T}(C_2); \omega : \texttt{skip end}\ \{r\}$, for some $\beta, \gamma, \omega \notin \mathit{FL}(C_1) \cup \mathit{FL}(C_2)$.

We get the following proof tree, where $\bar{\eta}$ stands for the environment $\eta[q_1/\beta, q_2/\gamma, q_3/\omega]$

$$\dfrac{p \Rightarrow q_1 \quad \Pi_2 \quad \dfrac{\vdots}{\bar{\eta} \vdash \{q_2\}\ \mathcal{T}(C_2)\ \{q_3\}} \quad \dfrac{q_3 \Rightarrow r}{\bar{\eta} \vdash \{q_3\}\ \texttt{skip}\ \{r\}}}{\eta \vdash \{p\}\ \texttt{begin}\,\beta : (\mathcal{T}(C_1\,[\gamma/\alpha]); \texttt{goto}\,\omega); \gamma : \mathcal{T}(C_2); \omega : \texttt{skip end}\ \{r\}}$$

where the proof tree Π_2 is

$$\dfrac{\dfrac{\vdots}{\bar{\eta} \vdash \{q_1\}\ \mathcal{T}(C_1\,[\gamma/\alpha])\ \{q\}} \quad \dfrac{q \Rightarrow q_3}{\bar{\eta} \vdash \{q\}\ \texttt{goto}\,\omega\ \{q_2\}}}{\bar{\eta} \vdash \{q_1\}\ \mathcal{T}(C_1\,[\gamma/\alpha]);\ \texttt{goto}\,\omega\ \{q_2\}}$$

By Lemma 3.5 $\bar{\eta} \vdash \{q_1\}\ \mathcal{T}(C_1\,[\gamma/\alpha])\ \{q\}$ iff $\eta\,[q_2/\alpha] \vdash \{q_1\}\ \mathcal{T}(C_1)\ \{q\}$, and $\bar{\eta} \vdash \{q_2\}\ \mathcal{T}(C_2)\ \{q_3\}$ iff $\eta \vdash \{q_2\}\ \mathcal{T}(C_2)\ \{q_3\}$.

Now, it is easy to see that $p \Rightarrow q_1$ and $\eta\,[q_2/\alpha] \vdash \{q_1\}\ \mathcal{T}(C_1)\ \{q\}$ and $q \Rightarrow q_3$ and $q_3 \Rightarrow r$, for some $q_1, q, q_3 \in \mathit{Cont}$, iff $\eta\,[q_2/\alpha] \vdash \{p\}\ \mathcal{T}(C_1)\ \{r\}$.

Hence, by inductive hypothesis, we get proofs for $\eta\,[q_2/\alpha] \vdash \{q_1\}\ C_1\ \{q\}$ and $\eta\,[q_2/\alpha] \vdash \{p\}\ \mathcal{T}(C_1)\ \{r\}$.

In summary, we get the proof tree

$$\dfrac{\dfrac{\vdots}{\eta\,[q_2/\alpha] \vdash \{p\}\ C_1\ \{r\}} \quad \dfrac{\vdots}{\eta \vdash \{q_2\}\ C_2\ \{r\}}}{\eta \vdash \{p\}\ \texttt{trap}\ \alpha\ \texttt{in}\ C_1\ \texttt{with}\ C_2\ \{r\}}$$

which is the same obtained by rule (Trap). □

4 Annotations as Pre-fixpoints

In this section we analyse the consequences of our point of view on a more practical issue, i.e. the problem of annotating a program in such a way that its correctness can be proved in a semi-automatic way.

First of all, talking about annotations requires to turn from the extensional approach we have taken until now (correctness assertions as triples $\{p\}\ C\ \{r\}$ with p, r predicates, i.e. semantic entities) to the intensional approach, where correctness assertions are triples $\{P\}\ C\ \{R\}$ with P, R formulas in some given language *Form*.

Analogously, we consider syntactic representations of continuation environments, ranged over by H, which are finite maps associating a formula with a label. We call them *formula environments*.

A formula P is semantically interpreted as a predicate $[\![P]\!] \in \mathit{Pred}$, and this interpretation naturally extends to formula environments. The validity of a correctness assertion is now defined by $H \models \{P\}\ C\ \{R\}$ iff $[\![H]\!] \models \{[\![P]\!]\}\ C\ \{[\![R]\!]\}$; the proof rules given in the preceding sections still work in the intensional version. Note that, anyway, the completeness of the proof system now holds only if the given language of formulas is expressive enough for representing all the valid assertions (see e.g. [12] for a detailed explanation of this point).

Coming now to annotations, we recall that they are formulas inserted in specific points in a program to indicate that some property is expected to hold there. Annotations can be used just for documentation purposes or as an help for proving the correctness of a program w.r.t. to a given specification. Indeed, the task of proving a correctness assertion $\{P\}\ C\ \{R\}$ can be in principle reduced to prove the validity of $P \Rightarrow \mathit{wlp}(C)R$; anyway, $\mathit{wlp}(C)R$ could be either non expressible in *Form* or expressible, but in a very convolute way (that is tipically the case when C is a command whose semantics is expressed by a fixpoint equation, like `while` and `block` commands in ***Goto***). In this case, the usual approach is to require the user to insert a certain number of annotations in C, getting an annotated command A; then, it is possible to extract from $\{P\}\ A\ \{R\}$ a set $\mathit{VC}(\{P\}\ A\ \{R\})$ of formulas in *Form*, called *verification conditions*, s.t. their validity guarantees the validity of the original assertion $\{P\}\ C\ \{R\}$. We illustrate the idea on an example, referring to [6, 12] for the formal definitions. The validity of the assertion $\{x = n\}\ \overline{C}\ \{y = n\}$, with n positive integer constant,

$$\overline{C} = \ y\ \texttt{:=}\ 0\ \texttt{;}\ \texttt{while}\ x \neq 0\ \texttt{do}\ (\ y\ \texttt{:=}\ y+1\ \texttt{;}\ x\ \texttt{:=}\ x-1\ )$$

can be established annotating $\overline{C}$ as follows

$$\overline{A} = \ y\ \texttt{:=}\ 0\ \texttt{;}\ \{x+y=n \wedge x \geq 0\}\ \texttt{while}\ x \neq 0\ \texttt{do}\ \{x+y=n \wedge x \geq 0\}\ (\ y\ \texttt{:=}\ y+1\ \texttt{;}\ x\ \texttt{:=}\ x-1\ ).$$

Indeed, the verification conditions $\mathit{VC}(\{x = n\}\ \overline{A}\ \{y = n\})$ turn out to be the formulas

$$x = n \Rightarrow x + 0 = n \wedge x \geq 0,$$
$$x + y = n \wedge x \geq 0 \wedge x \neq 0 \Rightarrow x + y = n \wedge x \geq 0,$$
$$(x + y = n \wedge x \geq 0) \wedge \neg(x \neq 0) \Rightarrow y = n$$

whose validity can be immediately proved.

Note that the task of choosing correct annotations is completely left to the user: one could wrongly annotate the program, and in this case obtain in the verification conditions some non valid formula.

Note moreover that the user is required to insert one annotation for each `while` command, written `while` B `do` $\{P\}\ C$ to stress that P is an invariant, and one annotation for each command sequence, written $C_1\,\texttt{;}\,\{P\}\ C_2$ (unless C_2 is a `skip` or an assignment command).

The aim of this section is to present a quite different approach to the problem of finding correct annotations, based on the intuition that there is a correspondence between annotations and labels in programs. We first explain this correspondence.

If a command C has a label, say α, then, as already pointed out, α denotes in the current environment η the continuation semantics (weakest precondition) of C w.r.t. the current continuation (postcondition). Hence a label can be always correctly replaced by its denotation.

Then, the problem of finding a set of annotations for a program sufficient for proving its correctness can be reduced to the problem of finding a tuple of formulas which denotes a pre-fixpoint of the functional which defines the continuation style semantics (weakest precondition) of the program.

Starting from this idea, we propose below an algorithm Gen for finding a set of annotations for a program which works quite differently from the traditional one.

First of all, we let the syntax below for *Form*.

$$P ::= \texttt{true} \mid \texttt{false} \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \mid \neg P \mid E_1 \; relop \; E_2 \mid X \mid P\,[E/I]$$

Note that in the language of formulas we allow *formula variables*, ranged over by X, taken in some denumerably infinite set $\mathcal{V}$, and formulas of the form $P\,[E/I]$, expressing the replacement of an identifier I by an expression E in a formula P. In the following we assume that formulas are always reduced in such a way that replacements are only of the form $X\,[E_1/I_1]\ldots[E_n/I_n]$ $(n \geq 0)$.

We denote by $Form(V)$ the set of the formulas with free formula variables in V, for $V \subseteq \mathcal{V}$. Moreover, we say that $\sigma : \mathcal{V} \to Form(\mathcal{V})$ is a (V,W)-*substitution* iff σ is the identity over $\mathcal{V} \setminus V$ and, for any $X \in V$, $\sigma(X) \in Form(W)$. We denote by $P\langle\sigma\rangle$ the result of applying the substitution σ to the formula P (modulo reduction of replacements explained above).

For any (V_1,W_1)-substitution σ_1 and (V_2,W_2)-substitution σ_2, the $(V_1 \cup\ V_2, (W_1 \setminus\ V_2) \cup W_2)$-substitution $\sigma_1;\sigma_2$ is defined by $(\sigma_1;\sigma_2)(X) = \sigma_1(X)\langle\sigma_2\rangle$ if $X \in V_1$, $\sigma_2(X)$ if $X \in V_2 \setminus V_1$.

Formula variables will be used in the algorithm to denote indeterminates in the fixpoint equation defining the continuation style semantics of a program.

The algorithm Gen takes in input four parameters: a formula environment H, a command C, a postcondition R and a finite set V of (already used) formula variables. The initial call on a closed (i.e. without free labels) command C and a closed (i.e. without free formula variables) postcondition R will be $\mathsf{Gen}(\emptyset, C, R, \emptyset)$. In a generic call, say $\mathsf{Gen}(H, C, R, V)$, C will only contain free labels which have an associated formula in H, and H and R will only contain free formula variables in V.

The algorithm on an initial call $\mathsf{Gen}(\emptyset, C, R, \emptyset)$, returns $Q_0, \mathcal{F}, A, U$, where $U = \{X_1, \ldots, X_n\}$ is a finite set of formula variables, $\mathcal{F}$ is a set of constraints $\{X_i \Rightarrow Q_i \mid i \in 1..n\}$, $Q_0, Q_1, \ldots, Q_n \in Form(U)$ and A is an annotated version of C where the annotations are $X_1, \ldots, X_n$.

The expected meaning is that, for any $(U,\emptyset)$-substitution $\sigma\{X_i \mapsto S_i \mid\ i \in\ 1..n\}$ "solution" of $\mathcal{F}$, i.e. s.t. $S_i \Rightarrow Q_i\langle\sigma\rangle$, for $i \in 1..n$, $Q_0\langle\sigma\rangle$ is a precondition for C w.r.t. R. Thus, given a correctness assertion $\{P\}\ C\ \{R\}$, we can conclude the validity of this assertion if, moreover, $P \Rightarrow Q_0\langle\sigma\rangle$.

In other words, the algorithm generates simultaneously (a schema of) an annotated version of C and the corresponding set of verification conditions, which contain free formula variables. Then a correctly annotated version of C is obtained for any instantiation of these variables which makes the verification conditions valid.

For instance, on the example above the algorithm returns the tuple:

- $X\,[0/y]$,
- $\{X \Rightarrow (X\,[y+1/y]\,[x-1/x] \wedge x \neq 0) \vee (y = n \wedge \neg(x \neq 0))\}$,
- y := 0 ; $\{X\}$ **while** $x \geq 0$ **do** (y := $y+1$; x := $x-1$),

- $\{X\}$.

Adding $P \Rightarrow Q_0\langle\sigma\rangle$, we get the following set of constraints

$$x = n \Rightarrow X\,[0/y],$$
$$X \wedge x \neq 0 \Rightarrow X\,[y+1/y]\,[x-1/x],$$
$$X \wedge \neg(x \neq 0) \Rightarrow y = n$$

which admits the easy solution $X = (x + y = n \wedge x \geq 0)$.

Note that, while in the traditional approach there are two steps, i.e. annotating the program (step left to the user) and producing the verification conditions from the annotated program (algorithmic step), our algorithm produces in parallel (the schema of) the annotated program and the verification conditions; the part left to the user is now finding a solution of the system of equations.

Moreover, the algorithm inserts an annotation only in any point where (keeping in mind the above explained correspondence between annotations and labels) the denotation of a label would be recursively defined (intuitively, it is possible to jump to this label from the continuation of the corresponding command).

On a generic call $\mathsf{Gen}(H, C, R, V)$, the algorithm returns $Q_0, \mathcal{F}, A, U$ like above with the difference that $\{X_1, \ldots, X_n\} = U \setminus V$ and σ is a $(U \setminus V, V)$-substitution. The meaning is consequently generalized (see Theorem 4.3 below).

The algorithm Gen is given in Fig. 4. We say that a command C is *well-formed* w.r.t. a formula environment H if $H(\alpha)$ is defined for all $\alpha \in FL(C)$. Given a substitution σ and a set of constraints $\mathcal{F} = \{P_i \Rightarrow Q_i \mid i \in 1..n\}$, we write $\sigma \models \mathcal{F}$ iff $P_i\langle\sigma\rangle \Rightarrow Q_i\langle\sigma\rangle$ is valid, for all $i \in 1..n$.

In order to prove the correctness of the algorithm (Theorem 4.3 below) we need the following fact and lemma.

Fact 4.1 *For any pair σ_1, σ_2 of substitutions, $\sigma_1;\sigma_2 \models \mathcal{F}$ iff $\sigma_2 \models \mathcal{F}\langle\sigma_1\rangle$.*

The lemma below formally expresses the fact that the result of the algorithm on a generic call is invariant modulo renaming of the (initially given) free formula variables.

Lemma 4.2 *Let H, C, R, V be a formula environment, a command, a formula and a set of formula variables s.t. C is well-formed w.r.t. H and $FV(H) \cup FV(R) \subseteq V$. Set $\mathsf{Gen}(H, C, R, V) = (Q_0, \mathcal{F}, A, U)$. Then, for any (V, W)-substitution σ s.t. $W \cap (U \setminus V) = \emptyset$,*

$$\mathsf{Gen}(H\langle\sigma\rangle, C, R\langle\sigma\rangle, W) = (Q_0\langle\sigma\rangle, \mathcal{F}\langle\sigma\rangle, A, W'),$$

where $W' = W \cup (U \setminus V)$.

Proof. By structural induction on C. □

Theorem 4.3 *Let H, C, R, V be as in Lemma 4.2 and P a formula s.t. $FV(P) \subseteq V$. Set $\mathsf{Gen}(H, C, R, V) = (Q_0, \mathcal{F}, A, U)$. Then, statments 1 and 2 are equivalent:*

1. *$H \vdash \{P\}\ C\ \{R\}$.*
2. *There exists a $(U \setminus V, V)$-substitution σ s.t. $\sigma \models \mathcal{F} \cup \{P \Rightarrow Q_0\}$.*

Proof. By structural induction on C. We show two cases (for brievety we omit the third component of the output, i.e. the annotated command, since it is not relevant for the thesis).

Seq We have $\mathsf{Gen}(H, C_1; C_2, R, V) = (Q_0, \mathcal{F}_1 \cup \mathcal{F}_2, U)$, if $\mathsf{Gen}(H, C_2, R, V) = (Q, \mathcal{F}_2, W)$ and $\mathsf{Gen}(H, C_1, Q, W) = (Q_0, \mathcal{F}_1, U)$.

To see that (1) implies (2), assume a proof tree for $H \vdash \{P\}\ C\ \{R\}$ is given. The last step must be an instantiation of the (Conc) rule. Therefore, there exists a formula Q' s.t. $H \vdash \{P\}\ C_1\ \{Q'\}$ and $H \vdash \{Q'\}\ C_2\ \{R\}$. By inductive hypothesis, there exists a $(W \setminus V, V)$-substitution σ_2 s.t. $\sigma_2 \models \mathcal{F}_2 \cup \{Q' \Rightarrow Q\}$. Hence, from rule (Conseq) we get also $H \vdash \{P\}\ C_1\ \{Q\langle\sigma_2\rangle\}$. From Lemma 4.2, $\mathsf{Gen}(H, C_1, Q\langle\sigma_2\rangle, V) = (Q_0\langle\sigma_2\rangle, \mathcal{F}_1\langle\sigma_2\rangle, U')$, where $U' = V \cup (U \setminus W)$. Now, by inductive hypothesis, there exists a $(U' \setminus V, V)$-substitution σ_1 s.t. $\sigma_1 \models \mathcal{F}_1\langle\sigma_2\rangle \cup \{P \Rightarrow Q\langle\sigma_2\rangle\}$. Now, since $U' \setminus V = U \setminus W$, and from Fact 4.1 it is clear that $\sigma_2;\sigma_1$ is a $(U \setminus V, V)$-substitution s.t. $\sigma_1;\sigma_2 \models \mathcal{F}_1 \cup \mathcal{F}_2 \cup \{P \Rightarrow Q_0\}$, as desired.

The converse implication can be proved analogously.

While We have $\mathsf{Gen}(H, \texttt{while}\ B\ \texttt{do}\ C, R, V) = (X, \mathcal{F} \cup \{X \wedge B \Rightarrow Q, X \wedge \neg B \Rightarrow R\}, U)$, if, for some $X \in \mathcal{V} \setminus V$, $\mathsf{Gen}(H, C, X, V \cup \{X\}) = (Q, F, U)$.

To see that (1) implies (2), assume given a proof tree for $H \vdash \{P\}\ \texttt{while}\ B\ \texttt{do}\ C\ \{R\}$. Analogously to the case above, we can conclude that there exists a formula I s.t. $H \vdash \{I \wedge B\}\ C\ \{I\}$ and the implications $P \Rightarrow I$ and $I \wedge \neg B \Rightarrow R$ hold. By inductive hypothesis, $H \vdash \{X \wedge B\}\ C\ \{X\}$ iff there exists a $(U \setminus V \setminus \{X\}, V \cup \{X\})$-substitution τ s.t. $\tau \models \mathcal{F} \cup \{X \wedge B \Rightarrow Q\}$. Then, it is clear that $\sigma = \tau;\{X \mapsto I\}$ is a $(U \setminus V, V)$-substitution s.t. $\sigma \models (\mathcal{F} \cup \{X \wedge B \Rightarrow Q, X \wedge \neg B \Rightarrow R\}) \cup (P \Rightarrow X)$.

The converse implication can be proved analogously. □

Note that a set of constraints $\mathcal{F} = \{X_i \Rightarrow Q_i \mid i \in 1..n\}$ represents a functional $\Phi : \mathit{Pred}^n \to \mathit{Pred}^n$. Hence, for any substitution $\sigma = \{X_i \mapsto S_i \mid i \in 1..n\}$ solution of $\mathcal{F}$ (i.e. $S_i \Rightarrow Q_i\langle\sigma\rangle$, $i \in 1..n$), the tuple $S_1, \ldots, S_n$ represents, since in *Pred* we have $p \Rightarrow r$ iff $p \geq r$, a pre-fixpoint of Φ. Hence $S_1, \ldots, S_n$ gives a set of correct annotations w.r.t. $\{P\}\ C\ \{R\}$ iff it represents a pre-fixpoint of Φ and, moreover, $P \Rightarrow Q_0\langle\sigma\rangle$.

5 Conclusion

We have shown that weakest precondition can be obtained as a particular instance of continuation style semantics taking as "answers", in the continuation style sense, booleans with a suitable order. Taking this approach, correctness assertions can be interpreted as inequalities over continuations and semantic clauses can be taken as starting point for constructing a proof system for a language. Moreover, the problem of finding a set of annotations sufficient for proving the correctness of a program w.r.t. a correctness assertion can be reduced to the problem of finding a tuple of assertions denoting a prefixed point of the functional defining the continuation style semantics of the program.

Note that, since the requirement that continuations are strict implies that non termination gives as answer the bottom element of booleans, the order we have chosen (*true* < *false*) leads to partial correctness (non terminating commands satisfy any postcondition). In order to get total correctness, it is enough to consider the inverse order (*false* < *true*), as it was in [7]. Anyway, in this case there is no natural derivation of proof rules from semantic clauses in the cases where the fixed point operator is used (for instance, the proof rule for the `while` command uses an ad-hoc side condition for ensuring termination).

The traditional approach to proof rules for languages with jumps uses a generalized form of postconditions, i.e. correctness assertions become of the form $\{p\}\ C\ \{\{\alpha_1 : q_1, \ldots, \alpha_n : q_n\}\}$ where $\alpha_1, \ldots, \alpha_n$ correspond to the possible kinds of termination of C (including a special label ν denoting "normal" termination), and $q_1, \ldots, q_n$ are the corresponding postconditions. This corresponds to the idea that usual correctness assertions are suitable for single-entry, single-exit structures, and the above form is the natural generalization to single-entry, multiple-exit structures [3, 9].

This point of view is very useful when we think in a "precondition-driven" way, i.e. we want to state, given a precondition p and a command C, which properties are expected to hold when the execution of C terminates, in some of the possible ways. Anyway, in this way there is no natural generalization of the notion of weakest precondition.

We take exactly the dual view (indeed, a correctness assertion like above corresponds in our proof systems to a judgment $\eta \vdash \{p\}\ C\ \{r\}$ where $\eta(\alpha_i) = q_i$ for $\alpha_i \neq \nu$, $r = q_i$ for $\alpha_i = \nu$): our approach is "postcondition-driven", i.e. we want to state, given a postcondition r and a command C, which property must hold before C and which properties must hold before executing the corresponding handler/command in any case of abnormal termination, in order that the postcondition r is guaranteed to hold after C.

This paper is intended to be a first step in analyzing the relationship between continuation style semantics and proof rules, and is mainly aimed at fixing the correspondence and showing its consequences in some well-known cases. Hence the interest is not in the results, which are standard, but in the new point of view over them. The continuation of our work will be the application of this point of view to cases where there is no clear idea of how a good proof system should be. In particular, we are interested in proof systems for the object oriented paradigm, which only recently have been the subject of some proposal [2, 8]. We think that our approach could help both in giving a cleaner view of what should be in this case an axiomatic semantics and in allowing a simple integration with the treatment of jumps, leading to proof systems suitable for object oriented languages with exception handling like Java.

References

[1] Special issue on continuations. *LISP and Symbolic Computation*, 6 and 7(3/4 and 1), 1993-1994.

[2] M. Abadi and K.R.M. Leino. A logic of object-oriented programs. In *TAPSOFT '97: Theory and Practice of Software Development*, number 1214 in Lecture Notes in Computer Science, pages 682–696. Springer Verlag, April 1997.

[3] M.A. Arbib and S. Alagić. Proof rules for `goto`s. *Acta Informatica*, 11:139–148, 1979.

[4] F. Cristian. Correct and robust programs. *IEEE Transactions on Software Engineering*, SE-10(2):163–174, 1984.

[5] E.W. Dijkstra. *A Discipline of Programming.* Prentice Hall, Englewood Cilffs, N.J., 1976.

[6] M.J.C. Gordon. *Programming Language Theory and its Implementation.* Prentice Hall, Englewood Cilffs, N.J., 1988.

[7] K. Jensen. Connection between Dijkstra's predicate transformers and denotational continuation semantics. Technical Report DAIMI PB-86, Computer Science Dept., Aarhus Univ., 1978.

[8] K.R.M. Leino. Ecstatic: An object-oriented programming language with an axiomatic semantics. In *4th Intl. Workshop on Foundations of Object Oriented Languages 1997*, 1997.

[9] K. Lodaya and R.K. Shyamasundar. Proof theory for exception handling in a tasking environment. *Acta Informatica*, 28:7–41, 1990.

[10] H.R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction.* John Wiley & Sons, 1992.

[11] D.A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn & Bacon, Inc., 1986.

[12] G. Winskel. *The Formal Semantics of Programming Languages - An Introduction.* Foundations of Computing Series. The MIT Press, Cambridge, Massachussets, 1993.

Input

- a formula environment H
- a command C well-formed w.r.t. H
- a postcondition R
- a finite set of variables V s.t. $FV(H) \cup FV(R) \subseteq V$

Output

- a formula Q_0 (a candidate precondition of C w.r.t. R under H)
- a finite set of constraints $\mathcal{F} = \{X_i \Rightarrow Q_i \mid i \in 1..n\}$
- an annotated version A of C where the annotations are $X_1, \ldots, X_n$
- a finite set U of formula variables ($U = V \cup \{X_1, \ldots, X_n\}$)

Algorithm $\mathsf{Gen}(H, C, R, V) = (Q_0, \mathcal{F}, A, U)$, where:

- $\mathsf{Gen}(H, \texttt{skip}, R, V) = (R, \emptyset, \texttt{skip}, V)$;
- $\mathsf{Gen}(H, I := E, R, V) = (R[E/I], \emptyset, I := E, V)$;
- $\mathsf{Gen}(H, C_1; C_2, R, V) = (Q_0, \mathcal{F}_1 \cup \mathcal{F}_2, A_1; A_2, U)$
  if $\mathsf{Gen}(H, C_2, R, V) = (Q, \mathcal{F}_2, A_2, W)$ and $\mathsf{Gen}(H, C_1, Q, W) = (Q_0, \mathcal{F}_1, A_1, U)$;
- $\mathsf{Gen}(H, \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, R, V) =$
  $(B \wedge P_1 \vee \neg B \wedge P_2, \mathcal{F}_1 \cup \mathcal{F}_2, \texttt{if } B \texttt{ then } A_1 \texttt{ else } A_2, U)$
  if $\mathsf{Gen}(H, C_1, R, V) = (P_1, \mathcal{F}_1, A_1, W)$ and $\mathsf{Gen}(H, C_2, R, W) = (P_2, \mathcal{F}_2, A_2, U)$
- $\mathsf{Gen}(H, \texttt{while } B \texttt{ do } C, R, V) =$
  $(X, \mathcal{F} \cup \{X \Rightarrow (Q \wedge B) \vee (R \wedge \neg B)\}, \{X\} \texttt{ while } B \texttt{ do } A, U)$
  if $\mathsf{Gen}(H, C, X, V \cup \{X\}) = (Q, F, A, U)$, for some $X \in \mathcal{V} \setminus V$;

- $\mathsf{Gen}(H, \texttt{raise}\,\alpha, R, V) = (H(\alpha), \emptyset, \texttt{raise}\,\alpha, V)$;
- $\mathsf{Gen}(H, \texttt{trap } \alpha \texttt{ in } C_1 \texttt{ with } C_2, R, V) = (Q_0, \mathcal{F}_1 \cup \mathcal{F}_2, \texttt{trap } \alpha \texttt{ in } A_1 \texttt{ with } A_2, U)$
  if $\mathsf{Gen}(H, C_2, R, V) = (Q, \mathcal{F}_2, A_2, W)$ and $\mathsf{Gen}(H[Q/\alpha], C_1, R, W) = (Q_0, \mathcal{F}_1, A_1, U)$;

- $\mathsf{Gen}(H, \texttt{goto}\,\alpha, R, V) = (H(\alpha), \emptyset, \texttt{goto}\,\alpha, V)$;
- $\mathsf{Gen}(H, \texttt{begin}\,\alpha_1 : C_1; \ldots; \alpha_n : C_n \texttt{ end}, R, V) =$
  $(X_1, \mathcal{F}, \texttt{begin}\{X_1\}\ \alpha_1 : A_1; \ldots; \{X_n\}\ \alpha_n : A_n \texttt{ end}, U)$
  if $\mathsf{Gen}(\overline{H}, C_i, X_{i+1}, V_i) = (Q_i, \mathcal{F}_i, A_i, V_{i+1})$ $(i < n)$ and $\mathsf{Gen}(\overline{H}, C_n, R, V_n) = (Q_n, \mathcal{F}_n, A_n, U)$,
  for some $X_1, \ldots, X_n \in \mathcal{V} \setminus V$, where
  $V_1 = V \cup \{X_1, \ldots, X_n\}$,
  $\overline{H} \equiv H[X_1/\alpha_1, \ldots, X_n/\alpha_n]$,
  $\mathcal{F} = \cup_{i=1}^{n} \mathcal{F}_i \cup \{X_i \Rightarrow Q_i\}$.

Figure 4: The Gen algorithm