Publ. RIMS, Kyoto Univ.
25 (1989), 415–489

Proofs and Programs: A Naïve Approach to Program Extraction

By

Hayao NAKAHARA*

Abstract

In this paper, we examine a naïve but precise relation between verification proofs of iterative programs with recursions and proofs in a predicate logic. For that purpose, a logic of total correctness of programs is introduced, which is an extension of a predicate logic and a variant of Pratt's dynamic logic equipped with program variables. Then it is shown that correctnes proofs of programs are regarded as proofs in a predicate logic. Conversely we show how programs and their correctness proofs can be obtained from proofs of their specifications.

These methods provide means to examine how difference among algorithms reflects difference among proofs and vice verse. Using them, we can replace processes of programming by processes of proving specifications. Our method does not depend on specific concrete theories, so we can use our method to problem solving on abstract domain.

§1. Introduction

1.1 Programming by proving specifications

In order to obtain reliability of programs, programming is roughly summed up as the following process:

(i) Describing a problem to be solved;
(ii) Constructing a program to solve the problem;
(iii) Demonstrating that the program solves the problem.

The third step is essential to guarantee the reliability of the program. For the sake of simplicity, we only consider problems to find results satisfying given properties under some conditions. In order to demonstrate that a program solves a problem rigorously, we need follow the above process in a formal way. The above process is reformulated as follows:

(i) Describing a formal specification for an informal problem;

Communicated by S. Takasu, November 2, 1988.
* Department of Information Science, Tokyo University, Tokyo 113, Japan,
Current address: Faculty of Integrated Arts and Sciences, Hiroshima University, Hiroshima 730, Japan.

(ii) Constructing a program for the specification;
(iii) Verifying formally that the program satisfies the specification.

To conduct this process, we must specify a formal system to describe specifications and specify a logic to describe how the specification language and programs are interrelated. We call the former the *underlying logic* and the latter the *program analysis logic*. Using the predicate logic as the underlying logic, a typical specification can be described as follows:

$$\forall \bar{x}.(\varphi[\bar{x}] \supset \exists \bar{y}.\psi[\bar{x}, \bar{y}]).$$

This expresses that:

For each input $\bar{x}$ satisfying the condition $\varphi[\bar{x}]$, find a result $\bar{y}$ satisfying the property $\psi[\bar{x}, \bar{y}]$.

In general, we cannot construct any program without understanding a problem. We usually analyze what properties are derived from the specification before constructing a program. However this analysis is closely related to a proof of correctness of the program obtained as a result of the analysis.

On the other hand, it is well-known that there is a close relation between a program and a constructive proof of its specification. If we can obtain a program from a proof of its specification, the above process is again reformulated as follows:

(i) Translating an informal problem to a formal specification;
(ii) Proving the specification;
(iii) Extracting a program from the proof.

In this approach, the correctness of the program with respect to its specification is guaranteed in the second step, and we need no separated verification proof. Moreover the third step may be easily mechanizable.

It seems unrealistic that we need a proof before constructing a program. However intensive analysis of a specification seems to be unavoidable, and if we need rigorous reliability of a program, a correctness proof is required in any case. Thereby this approach seems to be a shortcut to construct a flawless program, and it is important to establish the method for constructing a proof of a specification as easily as constructing a program.

1.2. Program analysis logic

It is desirable to treat specifications not only on specific concrete domains, but also on any abstract domains (or theories). Moreover it will ease a task of describing specification to utilize ordinary mathematics such as set theory, inductive definition and so forth. So the restriction on the underlying theory should be as small as possible. Hence we choose a usual first-order many sorted classical logic for our underlying logic.

In order to verify correctness of programs with respect to their specification, we must utilize some formal system. Our program analysis logic is a kind of logic of programs. We formulate it as an extension of our underlying logic. It has its root in Igarashi [11], Pratt's dynamic logic [16] and several variants of them [7], which originate in Hoare's system [10].

We shall describe some features of our logic.

For each program, we explicitly designate variables on which the program execution depends, and divide them into two classes, left variables and right variables. The pair of these two classes of variables is called the *p-type* of the program. Left variables of a program are the variables whose values may be changed during execution of the program. Right variables of a program are the variables whose values may be referred by the program. Using our program analysis logic, a specification for a *swap* program can be described as follows:

$$x=x_0 \wedge y=y_0 \supset \langle swap\colon (x, y;\ x, y)\rangle (x=y_0 \wedge y=x_0)$$

where *swap*: $(x, y; x, y)$ means *swap* has x and y as both left and right variables.

We have a new kind of formulas: for a program α and a formula φ, $\langle\alpha\rangle\varphi$ is a formula which express that α always terminates and φ always holds after the execution of α terminates. Thus our logic is a logic of total correctness.

Semantically, $\langle\alpha\rangle\varphi$ implies the existence of a state which satisfies φ. Thus $\langle\alpha\rangle$ acts like an existential quantifier. Actually if $\varphi \supset \langle\alpha\rangle\psi$ can be proved in a suitable subsystem of our program analysis logic, called a *construction part*, we can obtain a proof of $\varphi \supset \exists\bar{y}.\psi$ in the underlying logic where $\bar{y}$ are the left variables of α.

Moreover our logic permits use of variables ranging over programs (called *program variables*). These program variables enable us to treat recursive programs and they play the essential role in program extraction as described in the next section.

1.3. Program extraction

It is a well-known fact that mathematical proofs which are regarded as constructive in a certain sense relate to programs. In particular, existential quantifiers correspond to programs. For example, when a formula $\exists x.\varphi$ is proved by introducing $\exists$ quantifier, that is,

$$(*) \qquad \varphi_x[\tau] \rightarrow \exists x.\varphi,$$

we know τ is a value of x which satisfies φ. Thus we can extract a program fragment $x:=\tau$, and φ holds after the execution of $x:=\tau$. This fact is expressed by $\langle x:=\tau\rangle\varphi$ in our language. That is, from a proof of $(*)$ we can extract a proof of the following:

$$\varphi_x[\tau] \rightarrow \langle x:=\tau\rangle\varphi.$$

In this point of view, it is sufficient to proving this specification in

constructive way in order to obtain a program from a specification $\varphi \supset \exists x.\psi$.

However if we will correspond a program to every constructive proof, we should handle *higher-order* programs, in which functions and procedures are regarded as first class objects. Among previous works, Takasu [18] and Sato [17] use Gödel's functionals of finite types, T, and dialectica interpretation [6], Beeson [1], Hayashi [9] and Tatsuta [21] use Feferman's type free system and q-realizability interpretation [2, 4], Cornell [3] and Göteborg [8] groups use Martin-Löf's type theory [14].

These systems are based on the constructive logic, and not only existential quantifiers but also all of the other logical connectives, including disjunctions and implications, have a computational meaning. This is too complicated and hard to understand for novice programmers. It is desirable to treat proofs in a usual (non-constructive) theory and to give a computational meaning to only existential quantifiers.

However not all of the existential quantifiers are regarded as computable. Usual specifications have a lot of existential quantifiers not corresponding to programs. So we need to specify whether an existential quantifier correspond to a program or not. Lifschitz [12] proposes a logic which has both constructive and non-constructive existential quantifiers, but we take another approach.

All the systems listed above treat only functional programs, which have mathematically good properties. On the other hand, iterative programs play an important part in realistic programs. Hence it is also important to take usual iterative programs into consideration. So, in this paper, we establish a naïve but precise relationship between proofs and iterative programs. We only focus on *first-order* iterative programs with recursion. First-order means there is no facility to pass a program to another program through a parameter in execution time. As a result, the class of proofs which can be handled by our method is restricted. However most of realistic programs lie in this category, and the relation between proofs and programs becomes straightforward and easily understandable. So we can establish naïve method to extract a program from a proof of its specification.

Our extraction process of a program is as follows:

(i) We mark existential quantifiers corresponding to programs, denoted by $\tilde{\exists}$;

(ii) From a proof of

$$\tilde{\exists}\bar{x}.\psi \rightarrow \tilde{\exists}\bar{y}.\varphi,$$

we find a program α using a program variable ξ and construct a proof of

$$\langle\xi\rangle\psi \rightarrow \langle\alpha\rangle\varphi$$

in our program analysis logic.

Thus we not only extract a program but also construct a correctness proof of the program from the original proof.

Manna and Waldinger [13] also extract first-order programs in a classical logic framework. They however consider only functional programs and do not have non-constructive existential quantifiers. Our approach can handle a wider class of proofs than their approach and provide clearer insight into relationaship between proofs and programs.

Takasu [19, 20] extracts iterative programs from constructive proofs. However they can only handle well-behaved proofs which are built up from several *templates* of inference each of which is specific to a program construct.

1.4. Outline of this paper

In Section 2, we introduce a logical system $\mathbb{UL}\mathbb{0}$ for describing specifications and conducting mathematical proofs. $\mathbb{UL}\mathbb{0}$ is a usual first-order many sorted classical logic. We use a natural deduction style formulation, and $\mathbb{UL}\mathbb{0}$ is essentially same as $\mathbb{NK}$ of Gentzen [5, 15]. However $\mathbb{UL}\mathbb{0}$ treats partial terms as Beeson's $\mathbb{EON}$ [2]. Next, we modify $\mathbb{UL}\mathbb{0}$ and obtain $\mathbb{UL}\mathbb{1}$ so that we can easily extend it to a logic of programs.

In Section 3, we introduce a logic of programs $\mathbb{PL}\mathbb{i}$, the program analysis logic for iterative programs. $\mathbb{PL}\mathbb{i}$ is presented as an extension of $\mathbb{UL}\mathbb{1}$. In this paper, we only describe a formal system of $\mathbb{PL}\mathbb{i}$, and formal semantics of $\mathbb{PL}\mathbb{i}$ is not described. We also discuss that some class of proofs in $\mathbb{PL}\mathbb{i}$ can be regarded as proofs in an underlying logic.

In Section 4, we show Hoare Logic can be simulated in $\mathbb{PL}\mathbb{i}$ naturally. Hence $\mathbb{PL}\mathbb{i}$ has sufficient power to verify correctness of programs.

In Section 5, we state our main result, the program extraction theorem. Since not all the existential quantifiers are necessarily translated into programs, we introduce an auxiliary formal system $\tilde{\exists}$-$\mathbb{UL}\mathbb{1}$ as an extension of $\mathbb{UL}\mathbb{1}$. $\tilde{\exists}$-$\mathbb{UL}\mathbb{1}$ has a marked existential quantifier, $\tilde{\exists}$, to indicate which existential quantifier is to be translated into a program. In order to use $\mathbb{UL}\mathbb{0}$ as the underlying logic, we also give a formal system $\tilde{\exists}$-$\mathbb{UL}\mathbb{0}$ and show how proofs in $\tilde{\exists}$-$\mathbb{UL}\mathbb{0}$ can be transformed into proofs in $\mathbb{PL}\mathbb{i}$.

In Section 6, we extend our main theorem in order to naturally produce while programs. Then we discuss how while programs can be extracted and what programs can be extracted from proofs using ordinary mathematical induction.

In Section 7, We discuss some possible extensions of our method and give concluding remarks.

Notation. We write $\bar{\mathscr{E}}$ for the sequence of expressions $\mathscr{E}_1, \ldots, \mathscr{E}_n$. We also write $\bar{\mathscr{E}}$ for the finite set of expressions $\mathscr{E}_1, \ldots, \mathscr{E}_n$. In the context where a sequence notation $\bar{\mathscr{E}}$ appears, $\mathscr{E}_i$ denotes an element of $\bar{\mathscr{E}}$. We shall freely use *overline* notation to denote sequences and finite sets. For example, $\mathrm{f}(\bar{\tau})$ stands

for $\mathbb{f}(\tau_1, \ldots, \tau_n)$; for $2n$-ary infix predicate symbol $\prec$, $\bar{\sigma} \prec \bar{\tau}$ stands for $(\sigma_1, \ldots, \sigma_n) \prec (\tau_1, \ldots, \tau_n)$; $\overline{\mathbb{f}(\bar{\tau})}$ stands for $\mathbb{f}_1(\tau_{1,1}, \ldots, \tau_{1,m_1}), \ldots, \mathbb{f}_n(\tau_{n,1}, \ldots, \tau_{n,m_n})$; $x \in \bar{y}$ means x occurs in a sequence or a finite set $\bar{y}$; the phrase "*variables* $\bar{x}$ *do not occur in* $\mathscr{E}$" means no variable in a sequence or a finite set of variables $\bar{x}$ occurs in $\mathscr{E}$.

We use $\equiv$ for the syntactic equality and write $X - Y$ for the set difference of two sets X and Y.

§2. Underlying Logic

In this section, we introduce two underlying logic UL0 and UL1, both of which are many sorted first-order classical predicate logic with partial terms.

2.1. Underlying logic UL0

UL0 is essentially same as Gentzen's NK [5, 15].

Definition. Given a set of sorts $\mathscr{S}$. An *arity in* $\mathscr{S}$ is a pair of sequences of sorts $\bar{s}$ and $\bar{s}'$ in $\mathscr{S}$, denoted by $(\bar{s} \to \bar{s}')$, such that $length(\bar{s}') \leq 1$.

We call $\bar{s}$ (resp. $\bar{s}'$) the *domain* (resp. the *range*) of an arity of $(\bar{s} \to \bar{s}')$. An arity $(\bar{s} \to \bar{s}')$ is said to be an *object* arity, a *function* arity or a *predicate* arity if the domain $\bar{s}$ is empty, if both the domain $\bar{s}$ and range $\bar{s}'$ are not empty or if the range $\bar{s}'$ is empty, respectively.

Definition. A *language* $\mathscr{L}$ of UL0 consists of a nonempty set of sorts $\mathscr{S}_{\mathscr{L}}$ and a set of *nonlogical* symbols with each of which an arity in $\mathscr{S}_{\mathscr{L}}$ is associated.

A nonlogical symbol with an object arity is called an *object constant*, a nonlogical symbol with a function arity is called a *function constant* and a nonlogical symbol with a predicate arity is called a *predicate constant* of $\mathscr{L}$. $\mathscr{C}_{\mathscr{L}}$ (resp. $\mathscr{F}_{\mathscr{L}}$, $\mathscr{P}_{\mathscr{L}}$) denotes the set of object constants (resp. function constants, predicate constants) of $\mathscr{L}$. We assume that for each sort $s \in \mathscr{S}_{\mathscr{L}}$, a predicate constant $=$ with arity $(s, s \to)$ belongs to $\mathscr{P}_{\mathscr{L}}$. We use $=$ as an infix operator and $=$ represents the equality relation.

UL0 is designed to describe specifications of programs. For that purpose, we divide sorts (resp. nonlogical symbols) into two classes, *executable* and *nonexecutable* sorts (resp. symbols). Moreover, an executable part of a language $\mathscr{L}$ must form another language $e\mathscr{L}$, the *executable part* of $\mathscr{L}$. This implies the arity of each executable symbol lies in $\mathscr{S}_{e\mathscr{L}}$, namely executable sorts. Only the executable symbols are allowed to appear in programs.

From now on, we fix a language $\mathscr{L}$ and we shall often omit a subscript $\mathscr{L}$.

We assume a countable set of *variable symbols* is given. We divide variable symbols into two classes: *marked* and *unmarked* variable symbols. Our intention of marked and unmarked variables is that marked variables may occur in programs but unmarked variables may not.

Definition. A *variable* (or *object variable*) with arity $(\rightarrow s)$ is a pair of a variable symbol x and arity $(\rightarrow s)$, denoted by $x{:}(\rightarrow s)$ such that x is marked if and only if s is an executable sort. A variable with a marked (resp. unmarked) symbol is called *marked* (resp. *unmarked*) *variable.*

Later we simply write x or $x{:}s$ for $x{:}(\rightarrow s)$ unless it is necessary to indicate the sort of x explicitly. If we use a sequence notation $\bar{x}$ for variables, we assume all the variable symbol occurring in $\bar{x}$ are distinct. From now on, we assume all the variables having the same variable symbol have the same arity in terms, formulas, proofs and so on.

Definition. *Terms* of $\mathscr{L}$ and their *associated sort* in $\mathscr{S}_{\mathscr{L}}$ are inductively defined as follows:

- ○ a variable or a constant symbol with arity $(\rightarrow s)$ is a term of sort s;
- ⊙ if $\mathbf{f}$ is a function constant with arity $(s_1,\ldots, s_n \rightarrow s_0)$ and $\tau_1,\ldots, \tau_n$ are terms of sort $s_1,\ldots, s_n$ respectively, then $\mathbf{f}(\tau_1,\ldots, \tau_n)$ is a term of sort s_0.

Definition. Sequences of terms $\bar{\tau}$ and $\bar{\sigma}$ are *compatible* if $\mathit{length}(\bar{\tau}) = \mathit{length}(\bar{\sigma})$ and for each i, τ_i and σ_i have the same sort.

Definition. *Formulas* of $\mathscr{L}$ are inductively defined as follows:

- ○ $\bot$ is an (atomic) formula;
- ⊙ for each term τ, $\triangle\tau$ is an (atomic) formula;
- ⊙ if $\mathbb{p}$ is a predicate constant with arity $(s_1,\ldots, s_n \rightarrow)$ and $\tau_1,\ldots, \tau_n$ are terms of sort $s_1,\ldots, s_n$ respectively, then $\mathbb{p}(\tau_1,\ldots, \tau_n)$ is an (atomic) formula;
- ○ if $\prec$ is a predicate constant with arity $(s_1,\ldots, s_n, s_1,\ldots, s_n \rightarrow)$, $x_1,\ldots, x_n$ are distinct variables of sort $s_1,\ldots, s_n$, respectively, and χ is a formula then $\mathbb{TI}(\prec, \lambda x_1,\ldots, x_n.\chi)$ is an (atomic) formula;
- ⊙ if $\varphi_1,\ldots, \varphi_n$ are formulas, then $\varphi_1 \supset \varphi_2$, $\varphi_1 \wedge \ldots \wedge \varphi_n$ and $\varphi_1 \vee \ldots \vee \varphi_n$ are formulas;
- ⊙ if φ is a formula and $\bar{x}$ is a sequence of distinct variables, then $\forall\bar{x}.\varphi$ and $\exists\bar{x}.\varphi$ are formulas.

An atomic formula $\triangle\tau$ means that τ is *defined* or the computation of τ *terminates.* $\mathbb{TI}(\prec, \lambda\bar{x}.\chi)$ means that the relation $\prec$ admits *transfinite induction* or is a *well-founded* relation on $\{\bar{x} \mid \chi\}$.

We introduce several abbreviations: $\wedge_i\varphi_i$ stands for $\varphi_1 \wedge \ldots \wedge \varphi_n$; $\vee_i\varphi_i$ stands for $\varphi_1 \wedge \ldots \wedge \varphi_n$; $\bar{\tau} = \bar{\sigma}$ stands for $\wedge_i\tau_i = \sigma_i$; $\varphi \supset\subset \psi$ stands for $(\varphi \supset \psi) \wedge (\psi \supset \varphi)$; $\neg\varphi$ stands for $\varphi \supset \bot$; $\tau \neq \sigma$ stands for $\neg(\tau = \sigma)$; $\top$ stands for $\bot \supset \bot$; $\mathbb{TI}(\prec)$ stands for $\mathbb{TI}(\prec, \lambda\bar{x}.\top)$; for a set of formulas $\Gamma = \{\varphi_1,\ldots, \varphi_n\}$, $\wedge\Gamma$ stands for $\wedge_i\varphi_i$.

Free and *bound* occurrences of variables in a formula are defined as usual. All the occurrences of variables occurring in a term are free in the

term. For a term or a formula $\mathscr{E}$, $\mathrm{FV}\mathscr{E}$ denotes the set of free variables occurring in $\mathscr{E}$.

Definition. An *executable term* is a term such that all the sorts and nonlogical symbols occurring in it are executable. A *boolean expression* is a quantifier free formula such that all the sorts and nonlogical symbols in it are executable.

Definition. Terms $\bar{\tau}$ are said to be *substitutible for* $\bar{x}$ *in* φ if the following conditions hold:

(i) $\bar{\tau}$ and $\bar{x}$ are compatible;

(ii) each variable y occurring in $\bar{\tau}$ is not bound at each free occurrence of x_i in φ;

(iii) if x_i is marked then τ_i is executable.

In this case, we write $\varphi_{\bar{x}}[\bar{\tau}]$ for the expression obtained from φ by replacing each free occurrence of x_i by τ_i simultaneously.

Definition. Let $\mathscr{E}$ be a term or a formula, $\bar{x}$ and $\bar{u}$ be sequences of variables. We say that $\bar{x}$ are *renamable to* $\bar{u}$ *in* $\mathscr{E}$ if the following hold:

(i) $\bar{x}$ and $\bar{u}$ are compatible;

(ii) u_i is substitutible for x_i in $\mathscr{E}$;

(iii) if $u_i \in \mathrm{FV}\mathscr{E}$ then $u_i \in \bar{x}$;

(iv) x_i is marked if and only if u_i is marked.

If $\bar{x}$ and $\bar{u}$ are compatible variables, we write $\mathscr{E}^{\bar{u}}_{\bar{x}}$ to designate the expression obtained from a term or a formula $\mathscr{E}$ by replacing each free occurrence of x_i by u_i simultaneously. When we write $\mathscr{E}^{\bar{u}}_{\bar{x}}$, we assume $\bar{x}$ are renamable to $\bar{u}$ in $\mathscr{E}$.

We define $\mathrm{FV}\sigma$, $\sigma_{\bar{x}}[\bar{\tau}]$ and $\sigma^{\bar{u}}_{\bar{x}}$ for a term σ in the same way as formulas.

Now we describe the logical axioms and the inference rules in UL0. We adopt Gentzen's natural deduction style to describe our formal system [5, 15]. UL0 is a classical logic and contains the *law of excluded middle* as an axiom scheme.

Axioms and Rules 2.1. Axioms of UL0: Law of excluded middle, equality and definedness.

EM $\varphi \vee \neg \varphi$

Eq $x = x$

$x = y \supset y = x$

$x = y \wedge y = z \supset x = z$

$\bar{x}=\bar{y}\supset \mathbb{f}(\bar{x})=\mathbb{f}(\bar{y})$ $(\mathbb{f}\in\mathscr{F})$
$\bar{x}=\bar{y}\wedge \mathbb{p}(\bar{x})\supset \mathbb{p}(\bar{y})$ $(\mathbb{p}\in\mathscr{P})$

$\triangle$ $\triangle \mathbb{c}$ $(\mathbb{c}\in\mathscr{C})$
$\triangle x$
$\triangle \mathbb{f}(\bar{\tau})\supset \triangle\bar{\tau}$ $(\mathbb{f}\in\mathscr{F})$
$\mathbb{p}(\bar{\tau})\supset \triangle\bar{\tau}$ $(\mathbb{p}\in\mathscr{P})$
$\sigma=\tau\supset\triangle\tau$

Axioms and Rules 2.2. Inference rules of ULO are:

$\bot$ $\dfrac{\bot}{\varphi}$

$\wedge$-I $\dfrac{\varphi_1 \ \dots \ \varphi_n}{\wedge_i\varphi_i}$ $\wedge$-E $\dfrac{\wedge_i\varphi_i}{\varphi_k}$ (*for each k*)

$\vee$-I $\dfrac{\varphi_k}{\vee_i\varphi_i}$ (*for each k*) $\vee$-E $\dfrac{\vee_i\varphi_i \quad \begin{matrix}[\varphi_1]\\ \vdots\\ \psi\end{matrix} \ \dots \ \begin{matrix}[\varphi_n]\\ \vdots\\ \psi\end{matrix}}{\psi}$

$\supset$-I $\dfrac{\begin{matrix}[\varphi]\\ \vdots\\ \psi\end{matrix}}{\varphi\supset\psi}$ $\supset$-E $\dfrac{\varphi \quad \varphi\supset\psi}{\psi}$

$\forall$-I $\dfrac{\varphi^{\bar{a}}_{\bar{x}}}{\forall\bar{x}.\varphi}\langle\bar{a}\rangle$ $\forall$-E $\dfrac{\triangle\bar{\tau} \quad \forall\bar{x}.\varphi}{\varphi_{\bar{x}}[\bar{\tau}]}(*)$

$\exists$-I $\dfrac{\triangle\bar{\tau} \quad \varphi_{\bar{x}}[\bar{\tau}]}{\exists\bar{x}.\varphi}(*)$ $\exists$-E $\dfrac{\exists\bar{x}.\varphi \quad \begin{matrix}[\varphi^{\bar{a}}_{\bar{x}}]\\ \vdots\\ \psi\end{matrix}}{\psi}\langle\bar{a}\rangle$

$\mathrm{Ind}(\prec,\ \lambda\bar{x}.\chi)$ $\dfrac{\mathbb{TI}(\prec,\ \lambda\bar{x}.\chi) \quad \varphi^{\bar{a}}_{\bar{x}} \quad \begin{matrix}[\chi^{\bar{a}}_{\bar{x}},\ \forall\bar{x}.(\chi\wedge\bar{x}\prec\bar{a}\supset\varphi)]\\ \vdots\\ \chi\end{matrix}}{\forall\bar{x}.(\chi\supset\varphi)}\langle\bar{a}\rangle$

(*): for $\forall$-E and $\exists$-I, $\bar{\tau}$ must be substitutible for $\bar{x}$ in φ.

$\langle\bar{a}\rangle$: denotes that the *eigen variables* $\bar{a}$ must satisfy the usual *eigen variable condition.* That is, $\bar{a}$ must not occur free in the conclusion and in the assumptions (above the minor premise in the case of elimination rules) except explicitly mentioned formulas such as $\varphi^{\bar{a}}_{\bar{x}}$. In this case, we say that a_i is the eigen variable *for* x_i or a_i is the eigen variable *corresponding to* x_i. Note that we

do not exclude the case an eigen variable a_i corresponding to x_i is just x_i.

A bracketed formula occurring in a inference rule such as $[\varphi]$ in $\supset$-I is *discharged* from assumptions when the rule is applied. A discharged formula is also designated by a label attached on it. The attached label indicates which rule discharge the formula.

We remark the following points:

(i) In $\forall$-E and $\exists$-I, if x_i is marked then τ_i is executable;
(ii) In $\forall$-I, $\exists$-E and Ind $(\prec, \lambda\bar{x}.\chi)$, $\varphi^{\bar{a}}_{\bar{x}}$ and $\chi^{\bar{a}}_{\bar{x}}$ are subject to the tacit condition: $\bar{x}$ are renamable to $\bar{a}$ in φ and χ;
(iii) In particular, in $\forall$-I, $\exists$-E and Ind$(\prec, \lambda\bar{x}.\chi)$, x_i is marked if and only if a_i is marked.

We also remark that bracketed formulas need not be discharged. For example, in $\supset$-I, $\varphi\supset\psi$ can be inferred from ψ without discharging any φ from assumptions. In such a case we say that the discharged formula φ *is not used.* On the other hand, in the case where some occurrences of φ are actually discharged, we say that the discharged formula φ is *actually used.*

For each elimination rule, the premise which contains the logical symbol which is eliminated is called *a major premise* of the inference and the other premises are called *minor premises* of the inference.

Definition. A *sequent* is a pair of a set of formulas Γ and a formula φ, denoted by $\Gamma\to\varphi$. When Σ is a proof of φ from assumptions Γ in $\mathbb{UL0}$ then we write $\Sigma\vdash_{\mathrm{UL0}}\Gamma\to\varphi$. In this case, we say $\Gamma\to\varphi$ is *the sequent for* Σ. A *constituent sequent of* a proof Σ is the sequent of a subproof of Σ.

If φ has a proof from assumptions Γ, then we write $\vdash_{\mathrm{UL0}}\Gamma\to\varphi$, and we simply write $\vdash_{\mathrm{UL0}}\varphi$ when Γ is empty. When $\vdash_{\mathrm{UL0}}\varphi\to\psi$ and $\vdash_{\mathrm{UL0}}\psi\to\varphi$ hold, we write $\vdash_{\mathrm{UL0}}\varphi\leftrightarrow\psi$. We also write

$$\overset{\psi_1,\ldots,\psi_n}{\underset{\varphi}{\sum}}$$

for $\Sigma\vdash_{\mathrm{UL0}}\Gamma\to\varphi$, where $\{\psi_1,\ldots,\psi_n\}\subseteq\Gamma$.

Equality axioms have somewhat restricted form in $\mathbb{UL0}$, but a usual substitution property holds:

Proposition 2.3. (Substitution Lemma)

(i) *Suppose* $\bar{\tau}$ *are substitutible for* $\bar{x}$ *in* φ, *and* $\bar{a}$ *are fresh variables, then*

$$\vdash_{\mathrm{UL0}}\bar{a}=\bar{\tau}\wedge\varphi^{\bar{a}}_{\bar{x}}\leftrightarrow\bar{a}=\bar{\tau}\wedge\varphi_{\bar{x}}[\bar{\tau}].$$

(ii) *Suppose* $\bar{\tau}$ *and* $\bar{\tau}'$ *are substitutible for* $\bar{x}$ *in* φ. *Then*

$$\vdash_{\mathrm{UL0}}\bar{\tau}=\bar{\tau}',\ \varphi_{\bar{x}}[\bar{\tau}]\ \to\ \varphi_{\bar{x}}[\bar{\tau}'].$$

Definition. *Contexts* are formula-like expressions, which are built up from atomic formulas and special symbols $*_1, \ldots, *_n$ (n may be 0) and each $*_i$ has at most one occurrence.

We use an expression like $\mathscr{A}[*_1, \ldots, *_n]$ to denote a context. For a context $\mathscr{A}[*_1, \ldots, *_n]$ and formulas $\varphi_1, \ldots, \varphi_n$, $\mathscr{A}[\varphi_1, \ldots, \varphi_n]$ denotes the formula obtained from $\mathscr{A}[*_1, \ldots, *_n]$ by replacing each $*_i$ by φ_i. We also use a context to specify occurrences in a formula. We say that a variable x is *bound at* $*_j$ *in a context* $\mathscr{A}[\bar{*}]$ if $*_j$ lies in a scope of $\forall$ or $\exists$ which binds x. A variable x is *bound in a context* $\mathscr{A}[\bar{*}]$ if x is bound at some $*_i$ in $\mathscr{A}[\bar{*}]$. A term τ is *substitutible at* $*_j$ *in a context* $\mathscr{A}[\bar{*}]$ if the variables occurring free in τ are not bound at $*_j$ in $\mathscr{A}[\bar{*}]$, and is *substitutible in a context* $\mathscr{A}[\bar{*}]$ if τ is substitutible at some $*_j$ in $\mathscr{A}[\bar{*}]$. We shall often write $\mathscr{A}$ simply for $\mathscr{A}[*_1, \ldots, *_n]$. We adopt our substitution and renaming notations to contexts as those of formulas.

We give an example of a proof in $\mathbb{UL}\mathbb{0}$. In this example and all the examples described later, we assume that definedness of all terms holds and we do not explicitly express such a formula as $\triangle(a-1)$.

To exhibit formal proofs, we use the following notations:

$$\frac{\varphi_1 \quad \cdots \quad \varphi_n}{\psi}$$

means ψ is derived from $\varphi_1, \ldots, \varphi_n$ using several steps of inferences, and

$$\frac{\varphi}{\psi} \equiv$$

means $\varphi \equiv \psi$.

Example 1. Let $FACT$ be a set of axioms:

$$F(0, 1), \ \forall n, m.(F(n, m) \supset F(n+1, (n+1)*m)).$$

Executable symbols are $\{0, 1, +, -, \cdot, =\}$. We assume all the variables occurring in this example are marked variables. We shall exhibit a proof of $FACT \to \forall x.\exists z.F(x, z)$ in Peano arithmetic with $\mathbb{TI}(<)$ as an axiom (also denoted by PA).

First we construct several subproofs.

$$\prod \qquad \dfrac{\dfrac{\dfrac{}{a=0} \quad \dfrac{FACT}{F(0, 1)}}{F(a, 1)}\ \text{Substitution Lemma}}{\exists z.F(a, z)}\ \exists\text{-I}$$

$$\sum_1 \quad \dfrac{\dfrac{a \neq 0}{a-1<a}\ \text{PA} \qquad \dfrac{\forall x.(x<a \supset \exists z.F(x,z))}{a-1<a \supset \exists z.F(a-1,z)}\ \forall\text{-E}}{\exists z.F(a-1,z)}\ \supset\text{-E}$$

$$\sum_2 \quad \dfrac{\dfrac{\dfrac{a \neq 0}{a-1+1=a}\ \text{PA} \qquad \dfrac{F(a-1,b)}{F(a-1+1,(a-1+1)\cdot b)}\ FACT}{F(a, a\cdot b)}\ \text{Substitution Lemma}}{\exists z.F(a,z)}\ \exists\text{-I}$$

$$\sum \quad \dfrac{a \neq 0,\ \forall x.(x<a \supset \exists z.F(x,z)) \qquad \overset{(\exists\text{-E})}{a \neq 0,\ F(a-1,b)}}{\dfrac{\begin{matrix}\sum_1 \\ \exists z.F(a-1,z)\end{matrix} \qquad \begin{matrix}\sum_2 \\ \exists z.F(a,z)\end{matrix}}{\exists z.F(a,z).}\ \langle b\rangle,\exists\text{-E}}$$

Now let

$$IH \triangleq \forall x.(x<a \supset \exists z.F(x,z)).$$

Then a desired proof is the following:

$$\dfrac{\dfrac{}{\mathbb{T}\mathbb{I}(<)}\ \text{PA} \qquad \dfrac{\dfrac{}{a=0 \vee a \neq 0}\ \text{EM} \qquad \dfrac{\overset{(\vee\text{-E})}{a=0} \qquad \overset{(\vee\text{-E})\ (\text{IND})}{a \neq 0,\ IH}}{\begin{matrix}\prod \\ \exists z.F(a,z)\end{matrix} \qquad \begin{matrix}\sum \\ \exists z.F(a,z)\end{matrix}}\ \vee\text{E}}{\exists z.F(a,z)}\ \langle a\rangle,\text{IND}}{\forall x.\exists z.F(x,z).}$$

2.2. Underlying logic UL1

The concept of substitution is used to describe UL0. However defining substitution for programs is slightly difficult. For example, consider the following program fragment;

$$(\text{if } z>0 \text{ then } x:=y \text{ else } y:=x);\ z:=x.$$

It is easily seen that substitution for the first occurrence of x is impossible and substitution for the second occurrence of x is possible, but we can not decide whether substitution for the third occurrence of x is possible or not. Thus it is hard to define the concept of substitution for programs.

In order to avoid this difficulty, we introduce a new quantifier let to UL0, and we obtain a formal system UL1. Inference rules which are described using substitution in UL0, that is $\forall$-E and $\exists$-I rules, are redefined using let quantifiers in UL1.

Difinition. *Formulas* of UL1 are inductively defined as those of UL0 with the following new clause:

- if φ is a formula, $\bar{x}$ are variables, $\bar{\tau}$ are terms compatible with $\bar{x}$, and τ_i is executable when x_i is marked, then **let** $\bar{x} \leftarrow \bar{\tau}$ **in** φ is a formula.

The meaning of **let** $\bar{x} \leftarrow \bar{\tau}$ **in** φ is the meaning of φ under the condition that the values of $\bar{x}$ are set to the values of $\bar{\tau}$. In **let** $\bar{x} \leftarrow \bar{\tau}$ **in** φ, we let $\bar{x}$ become bound but the variables occurring in $\bar{\tau}$ be free.

Axioms and Rules 2.4. Inference rules for **let** quantifiers are:

$$\textbf{let-I}\quad \frac{\begin{array}{c}[\bar{a}=\bar{\tau}]\\ \vdots\\ \triangle\bar{\tau} \quad \varphi^{\bar{a}}_{\bar{x}}\end{array}}{\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi}\langle\bar{a}\rangle \qquad \textbf{let-E}\quad \frac{\begin{array}{c}[\bar{a}=\bar{\tau},\varphi^{\bar{a}}_{\bar{x}}]\\ \vdots\\ \textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi \quad \psi\end{array}}{\psi}\langle\bar{a}\rangle$$

$\forall$-E and $\exists$-I become:

$$\forall\text{-E}\quad \frac{\triangle\bar{\tau}\ \ \forall\bar{x}.\varphi}{\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi} \qquad \exists\text{-I}\quad \frac{\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi}{\exists\bar{x}.\varphi}$$

Proposition 2.5. (Renaming Variables) *Let φ be a* UL1 *formula, $\bar{a}$ and $\bar{b}$ be sequences of distinct variables. If $\bar{a}$ are renamable to $\bar{b}$ in φ, then*

$$\vdash_{\text{UL1}} \bar{a}=\bar{b},\ \varphi \rightarrow \varphi^{\bar{b}}_{\bar{a}}$$

Proposition 2.6. (Substitution Lemma for UL1) *Substitution Lemma for* UL0 (*Proposition* 2.3) *also holds for* UL1.

In particular, if $\bar{\tau}$ are substitutible for $\bar{x}$ in φ, then

$$\vdash_{\text{UL1}} \textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi \leftrightarrow \triangle\bar{\tau}\wedge\varphi_{\bar{x}}[\bar{\tau}].$$

Proposition 2.7. UL1 *is conservative over* UL0. *That is, for a set of* UL0 *formulas Γ and a* UL0 *formula φ,*

$$\vdash_{\text{UL0}} \Gamma \rightarrow \varphi \text{ if and only if } \vdash_{\text{UL1}} \Gamma \rightarrow \varphi.$$

By this proposition, in UL1, we can use UL0 specific inference rules as derived rules. In order to avoid confusion, $\forall$-E (resp. $\exists$-I) of the form in UL0 is called $\forall$-E_0 (resp. $\exists$-I_0).

We can extend the notion of renaming and substitution to proofs. We write $\sum^{\bar{a}}_{\bar{x}}$ to denote the proof like figure obtained from Σ by replacing all the free occurrences of $\bar{x}$ by $\bar{a}$ and write $\sum_{\bar{x}}[\bar{\tau}]$ to denote the proof like figure obtained from Σ by substituting $\bar{\tau}$ for $\bar{x}$.

Proposition 2.8. *If Σ is a proof and $\bar{a}$ are variables not occurring in Σ, then $\sum^{\bar{a}}_{\bar{x}}$ is also a proof.*

Proposition 2.9. *If Σ is a proof, each variable in $\bar{x}$ does not occur as an eigen variable in Σ, and $\bar{\tau}$ are terms substitutible for all the free occurrences of $\bar{x}$ in Σ, then $\sum_{\bar{x}}[\bar{\tau}]$ is also a proof.*

Definition. We say that φ' is a *variant* of φ if φ' can be obtained from φ by a sequence of replacements of the following type: replace a part $\mathbb{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbb{in}\,\psi$ (resp. $\forall\bar{x}.\psi$, $\exists\bar{x}.\psi$) by $\mathbb{let}\,\bar{y}\leftarrow\bar{\tau}\,\mathbb{in}\,\psi^{\bar{y}}_{\bar{x}}$ (resp. $\forall\bar{y}.\psi^{\bar{y}}_{\bar{x}}$, $\exists\bar{y}.\psi^{\bar{y}}_{\bar{x}}$), where $\bar{y}$ are variables not free in ψ and $\bar{x}$ are renamable to $\bar{y}$ in φ.

Proposition 2.10. (Variant Lemma) *If φ' is a variant of φ, then*

$$\vdash_{\mathrm{ULI}} \varphi \leftrightarrow \varphi'.$$

Definition. A formula satisfies the *condition for variables* ($\mathbb{CV}$) if

- No variables occur both free and bound;
- Any bound variable in a subformula must not be bound again.

A context $\mathscr{A}[\bar{*}]$ satisfies $\mathbb{CV}$ if $\mathscr{A}[\overline{\perp}]$ satisfies $\mathbb{CV}$.

A sequent or a proof satisfies the *condition for variables* ($\mathbb{CV}$) if

- No variables occur both free and bound;
- All the formulas occurring in it satisfy $\mathbb{CV}$;
- For a proof, all the eigen variables are distinct.

If only marked variables are considered, they are said to satisfy $\mathbb{CV}$ *for marked variables.*

For example, $\forall x.(\varphi \supset \exists x.\psi)$ does not enjoy $\mathbb{CV}$.

From Variant Lemma, there is no loss of generality by restricting proofs to those satisfying $\mathbb{CV}$ ([15]).

Proposition 2.11.

(i) *For any formula φ there exists a formula ψ satisfying $\mathbb{CV}$ and*

$$\vdash_{\mathrm{ULI}} \varphi \leftrightarrow \psi.$$

(ii) *If*

$$\sum \vdash_{\mathrm{ULI}} \Gamma \to \varphi,$$

then there exist variants Γ'(resp. φ') of Γ(resp. φ) and a proof $\sum'$ satisfying $\mathbb{CV}$ such that

$$\sum' \vdash_{\mathrm{ULI}} \Gamma' \to \varphi'.$$

§3. Program Analysis Logic PL

We introduce a logic called *a program analysis logic* PL, to analyze programs'

behavior. We construct $\mathbb{PL}$ as an extension of an underlying logic $\mathbb{UL}0$ or $\mathbb{UL}1$. In particular, we introduce a logic for iterative programs (or Algol-like programs). We call this logic *program analysis logic for iterative programs with recursion* $\mathbb{PL}i$.

We do not pursue expressibility and completeness issues of our formal system, but we pay attention to the following points:

- straightforward relationship between programs and proofs;
- easiness of verifying programs.

In this paper, we do not present the formal semantics of our programming language and our logic $\mathbb{PL}i$. The intuitive meaning of programs and formulas of $\mathbb{PL}i$ is considerably obvious. We only mention that our axioms and rules are sound to an intuitive semantics of our system.

3.1. Program analysis logic $\mathbb{PL}i$

Definition. Let $\bar{x}$ and $\bar{y}$ be finite sets of marked variables of executable sorts. A *program type*, simply called a p-type, is a pair of $\bar{x}$ and $\bar{y}$, $(\bar{x}; \bar{y})$.

For a p-type $(\bar{x}; \bar{y})$, each element of $\bar{x}$ (resp. $\bar{y}$) is called a *left variable* (resp. a *right variable*). Later we shall define a p-type for each program α. Then we let $\mathbb{LV}\alpha$ (resp. $\mathbb{RV}\alpha$) denote the set of left variables (resp. right variables) of the p-type of α. A right variable of a program α is a variable whose value may affect an execution of α, and a left variable of α is a variable whose value may be affected by the execution of α.

A left variable which is not a rigth variable is called a *pure left variable*, and $\mathbb{pLV}\alpha$ denotes the set of the pure left variables of α.

Definition. Two p-types $(\bar{x}; \bar{y})$ and $(\bar{u}; \bar{v})$ are said to be *compatible* if the $(\bar{u}; \bar{v})$ is obtained by renaming of variable symbols from $(\bar{x}; \bar{y})$.

We assume a countable set of *program variable symbols* is given.

Definition. A *program variable* consists of a program variable symbol ξ and a p-type $(\bar{x}; \bar{y})$, denoted by $\xi:(\bar{x}; \bar{y})$.

Definition. Two program variables with the same program variable symbol, $\xi:(\bar{x}; \bar{y})$ and $\xi:(\bar{u}; \bar{v})$, are said to be *compatible* if $(\bar{x}; \bar{y})$ and $(\bar{u}; \bar{v})$ are compatible.

Definition. *Programs* and their associated p-type are defined as follows:

- a program variable $\xi:(\bar{x}; \bar{y})$ is an (atomic) program with p-type $(\bar{x}; \bar{y})$;
- if $\bar{x} \subseteq \bar{y}$ then $\mathtt{null}:(\bar{x}; \bar{y})$ is an (atomic) program with p-type $(\bar{x}; \bar{y})$;
- if $\bar{x}$ are marked variables and $\bar{\tau}$ are executable terms compatible with $\bar{x}$, then $\bar{x} := \bar{\tau}$ is an (atomic) program with p-type $(x_1, \ldots, x_n; \bigcup_i \mathbb{FV}\tau_i)$;

- ⊙ if α and β are programs, then $\alpha;\beta$ is a program with *p-type* $(\mathbb{LV}\alpha \cup \mathbb{LV}\beta;\ \mathbb{RV}\alpha \cup (\mathbb{RV}\beta - \mathbb{LV}\alpha))$;
- ⊙ if $\pi_1,\ldots,\ \pi_n$ are boolean expressions and $\alpha_1,\ldots,\ \alpha_n$ are programs, then $(\pi_1 \to \alpha_1 | \cdots | \pi_n \to \alpha_n)$ is a program with p-type $(\bigcup_i \mathbb{LV}\alpha_i;\ \bigcup_i \mathbb{FV}\pi_i \cup \bigcup_i \mathbb{RV}\alpha_i \cup \bigcup_i ((\bigcup_j \mathbb{LV}\alpha_j) - \mathbb{LV}\alpha_i))$;
- ○ if $\xi:(\bar{x};\ \bar{y})$ is a program variable and α is a program with p-type $(\bar{x};\ \bar{y})$ in which all the occurrences of program variables with the program variable symbol ξ are compatible with $\xi:(\bar{x};\ \bar{y})$, then $\mu\xi:(\bar{x};\ \bar{y}).\alpha$ is a program with p-type $(\bar{x};\ \bar{y})$;
- ○ if $\bar{x}$ are marked variables, $\bar{\tau}$ are executable terms compatible with $\bar{x}$ and α is a program, then $\mathbb{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbb{in}\ \alpha$ is a program with p-type $(\mathbb{LV}\alpha - \{x_1,\ldots,\ x_n\};\ \bigcup_i \mathbb{FV}\tau_i \cup (\mathbb{RV}\alpha - \{x_1,\ldots,\ x_n\}))$;
- ○ if $\bar{x}$ are marked variables and α is a program, then $\mathbb{local}\ \bar{x}\ \mathbb{in}\ \alpha$ is a program with p-type $(\mathbb{LV}\alpha - \{x_1,\ldots,\ x_n\};\ \mathbb{RV}\alpha - \{x_1,\ldots,\ x_n\})$;
- ○ if $\bar{x}$ are marked variables and α is a program, then $\mathbb{var}\ \bar{x}\ \mathbb{in}\ \alpha$ is a program with p-type $(\mathbb{LV}\alpha \cup \{x_1,\ldots,\ x_n\};\ \mathbb{RV}\alpha - \{x_1,\ldots x_n\})$.

A program variable symbol ξ is bound in $\mu\xi:(\bar{x};\ \bar{y}).\alpha$. Free and bound occurrences of program variables are defined as usual. We shall often omit the p-type $(\bar{x};\ \bar{y})$ of $\xi:(\bar{x};\ \bar{y})$ and write simply ξ.

We briefly explain the meanings of each program construct:

(i) $\mathbb{null}:(\bar{x};\ \bar{y})$ is a program which has no effects;
(ii) $\bar{x}:=\bar{\tau}$ is an assignment statement;
(iii) $\alpha;\beta$ is a sequential composition of α and β;
(iv) $(\pi_1 \to \alpha_1 | \cdots | \pi_n \to \alpha_n)$ is a nondeterministic choice statement;
(v) $\mu\xi:(\bar{x};\ \bar{y}).\alpha$ introduces a recursion;
(vi) $\mathbb{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbb{in}\ \alpha$ makes $\bar{x}$ local in α and initial values of $\bar{x}$ is set to the values of $\bar{\tau}$;
(vii) $\mathbb{local}\ \bar{x}\ \mathbb{in}\ \alpha$ makes $\bar{x}$ local in α, but initial values of $\bar{x}$ are not specified;
(viii) $\mathbb{var}\ \bar{x}\ \mathbb{in}\ \alpha$ initializes $\bar{x}$ before the execution of α, but the values of $\bar{x}$ are not specified.

For $\mathbb{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbb{in}\ \alpha$ and $\mathbb{local}\ \bar{x}\ \mathbb{in}\ \alpha$, variables $\bar{x}$ (said to be *local* variables) are invisible after execution of these statements, but $\bar{x}$ are visible after execution of $\mathbb{var}\ \bar{x}\ \mathbb{in}\ \alpha$, hence $\bar{x}$ are bound in $\mathbb{let}$ and $\mathbb{local}$ programs. We however consider $\bar{x}$ in $\mathbb{var}\ \bar{x}\ \mathbb{in}\ \alpha$ are not bound in the program. A $\mathbb{var}$ construct is not essential but is included by a technical reason.

We write $\alpha:(\bar{x};\ \bar{y})$ if α has p-type $(\bar{x};\ \bar{y})$.

Example 2. The following *fact* is a factorial program which computes the factorial of x and return its value in z.

$$fact \triangleq \mu\xi:(z;\ x).(x=0 \to z:=1 \mid x \neq 0 \to (\mathbb{let}\ x \leftarrow x-1\ \mathbb{in}\ \xi:(z;\ x));\ z:=z\cdot x).$$

The p-types of subprograms of *fact* are:

- $z := 1 : (z;)$
- $(\mathbb{let}\ x \leftarrow x - 1\ \mathbb{in}\ \xi:(z; x)) : (z; x)$
- $z := z \cdot x : (z; x, z)$
- $((\mathbb{let}\ x \leftarrow x - 1\ \mathbb{in}\ \xi:(z; x)); z := z \cdot x) : (z; x)$
- $(x = 0 \rightarrow z := 1 \mid x \neq 0 \rightarrow (\mathbb{let}\ x \leftarrow x - 1\ \mathbb{in}\ \xi:(z; x)); z := z \cdot x) : (z; x).$

Hence the p-type of *fact* is $(z; x)$.

We need to substitute programs for program variables to describe $\mathbb{PL}\mathrm{i}$.

Definition. A program α with p-type $(\bar{u}; \bar{v})$ is *substitutible for a program variable* $\xi:(\bar{x}; \bar{y})$ if $\bar{u} \subseteq \bar{x}$ and $\bar{v} \subseteq \bar{y}$.

Definition. Let α be a program with p-type $(\bar{x}; \bar{y})$, $\bar{u}$ be compatible with $\bar{x}$, and $\bar{v}$ be compatible with $\bar{y}$. Moreover we assume that x_i is identical to y_j if and only if the corresponding u_i is identical to v_j. Then we write $\alpha_{\bar{x};\bar{y}}[\bar{u}; \bar{v}]$ for the program obtained from α by renaming $\bar{x}$ and $\bar{y}$ to $\bar{u}$ and $\bar{v}$, respectively.

Definition. Let a program α be substitutible for a program variable $\xi:(\bar{x}; \bar{y})$. We can extend the p-type of α to $(\bar{x}; \bar{y})$ by composing $\mathbb{null}$ program with appropriate p-type to α, and let α' be such a program, $\mathbb{null}; \alpha$. Then $\beta_{\xi}[\alpha]$ is the program obtained from a program β by replacing each free occurrence of $\xi:(\bar{u}; \bar{v})$ by $\alpha'_{\bar{x};\bar{y}}[\bar{u}; \bar{v}]$.

We are now in the position to state formulas, axioms and inference rules for $\mathbb{PL}\mathrm{i}$.

First we assume that our language has a constant symbol ι_s for each sort s.

Definition. *Formulas* of $\mathbb{PL}\mathrm{i}$ are inductively defined as those of $\mathbb{UL}1$ with the following new clauses:

- if π is a boolean expression, then $\triangle\pi$ is a formula;
- if α is a program and φ is a formula, then $\langle\alpha\rangle\varphi$ is a formula.

Our intention of $\langle\alpha\rangle\varphi$ differs from one of Pratt's [16]. We interpret this formula as follows: α always terminates and φ always holds after the execution of α terminates. That is, $\mathbb{PL}\mathrm{i}$ is a logic of total correctness.

A formula is said to be *program-free* if no subformulas of the form $\langle\alpha\rangle\psi$ occur in it. For program variables we impose the following condition: all the program variables having the same program variable symbol must be compatible in every formula.

In programs, we consider that an (object) variable is bound only by $\mathbb{let}$ and $\mathbb{local}$ quantifiers. Hence $\mathbb{FV}\langle\alpha\rangle\varphi = \mathbb{FV}\varphi \cup \mathbb{LV}\alpha \cup \mathbb{RV}\alpha$. We also define $\mathbb{FV}\alpha = \mathbb{LV}\alpha \cup \mathbb{RV}\alpha$.

We now describe axioms and inference rules of $\mathbb{PL}i$. The axioms and inference rules of $\mathbb{UL}1$ are also those of $\mathbb{PL}i$. The axioms and inference rules specific to $\mathbb{PL}i$ are listed below. An asterisk (*) which is attached to names of some rules indicates the rule is not included in the construction part which is defined in the Section 3.2.

Axioms and Rules 3.1. Renaming rule and axioms for boolean expressions:

Rename $\dfrac{\bar{a}=\bar{b}\ \varphi}{\varphi^{\bar{b}}_{\bar{a}}}$ if $\bar{a}$ is renamable to $\bar{b}$ in φ

Boolean $\triangle \wedge_i \pi_i \supset \subset \wedge_i \triangle \pi_i$
$\triangle \vee_i \pi_i \supset \subset \wedge_i \triangle \pi_i$
$\triangle(\pi \supset \pi') \supset \subset \triangle\pi \wedge \triangle\pi'$
$\triangle \bot$

Axioms and Rules 3.2. General rules for programs:

$$\langle\alpha\rangle\text{-rule}\quad \frac{\langle\alpha\rangle\varphi \qquad \begin{matrix}[\varphi^{\bar{a}}_{\bar{x}}]\\ \vdots \\ \psi^{\bar{a}}_{\bar{x}}\end{matrix}}{\langle\alpha\rangle\psi}\langle\bar{a}\rangle \qquad \text{where } \bar{x}=\mathbb{LV}\,\alpha$$

In $\langle\alpha\rangle$-rule, the premise $\langle\alpha\rangle\varphi$ is called the major premise and $\psi^{\bar{a}}_{x}$ is called the minor premise just like the case for elimination rules. The eigen variable condition for $\langle\alpha\rangle$-rule is slightly different from one for other rules: the eigen variables $\bar{a}$ may occur in the assumptions above the major premise and also in α.

$\langle\alpha\rangle$-E $\dfrac{\langle\alpha\rangle\varphi}{\varphi}$ if $\mathbb{LV}\alpha \cap \mathbb{FV}\varphi = \phi$

$\forall$-$\langle\alpha\rangle$ $\dfrac{\langle\alpha\rangle\varphi}{\forall\bar{x}.\langle\alpha\rangle\varphi}$ if $\bar{x} \subseteq \mathbb{pLV}\alpha$

$\langle\alpha\rangle$-$\supset$* $\dfrac{\langle\alpha\rangle(\varphi \supset \psi)\ \langle\alpha\rangle\varphi}{\langle\alpha\rangle\psi}$

$\langle\alpha\rangle$-$\wedge$* $\dfrac{\langle\alpha\rangle\varphi_1 \cdots \langle\alpha\rangle\varphi_n}{\langle\alpha\rangle\wedge_i\varphi_i}$ $\langle\alpha\rangle$-$\forall$* $\dfrac{\forall\bar{x}.\langle\alpha\rangle\varphi}{\langle\alpha\rangle\forall\bar{x}.\varphi}(\bar{x}\notin\mathbb{LV}\alpha \cup \mathbb{RV}\alpha)$

Exch $\dfrac{\langle\alpha\rangle\langle\beta\rangle\varphi}{\langle\beta\rangle\langle\alpha\rangle\varphi}$ *if* $\mathbb{LV}\alpha\cap\mathbb{RV}\beta=\phi$, $\mathbb{LV}\beta\cap\mathbb{RV}\alpha=\phi$

If we have no program variables, then these rules are derived from other axioms and rules for each program construct (described below). However if program variables are presented, these rules are essential.

Axioms and Rules 3.3. Axioms and rules for each program construct:

$$\langle\textbf{let}\rangle\text{-I}\quad \frac{\triangle\bar{\tau}\quad \langle\alpha^{\bar{a}}_{\bar{x}}\rangle\varphi}{\langle\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \alpha\rangle\varphi}\langle\bar{a}\rangle \qquad \langle\textbf{let}\rangle\text{-E}\quad \frac{\langle\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \alpha\rangle\varphi \qquad \begin{matrix}[\bar{a}=\bar{\tau},\ \langle\alpha^{\bar{a}}_{\bar{x}}\rangle\varphi]\\ \vdots\\ \psi\end{matrix}}{\psi}\langle\bar{a}\rangle$$

$$\langle\textbf{local}\rangle\text{-I}\quad \frac{\langle\textbf{let}\ \bar{x}\leftarrow\bar{\imath}\ \textbf{in}\ \alpha\rangle\varphi}{\langle\textbf{local}\ \bar{x}\ \textbf{in}\ \alpha\rangle\varphi} \qquad \langle\textbf{local}\rangle\text{-E}\quad \frac{\triangle\bar{\tau}\ \langle\textbf{local}\ \bar{x}\ \textbf{in}\ \alpha\rangle\varphi}{\langle\textbf{let}\ \bar{x}\leftarrow\bar{\imath}\ \textbf{in}\ \alpha\rangle\varphi}$$

$$\langle\textbf{var}\rangle\text{-I}\quad \frac{\textbf{let}\ \bar{x}\leftarrow\bar{\imath}\ \textbf{in}\ \langle\alpha\rangle\varphi}{\langle\textbf{var}\ \bar{x}\ \textbf{in}\ \alpha\rangle\varphi} \qquad \langle\textbf{var}\rangle\text{-E*}\quad \frac{\langle\textbf{var}\ \bar{x}\ \textbf{in}\ \alpha\rangle\varphi}{\textbf{let}\ \bar{x}\leftarrow\bar{\imath}\ \textbf{in}\ \langle\alpha\rangle\varphi}$$

$$\langle\textbf{null}\rangle\text{-I}\quad \frac{\varphi}{\langle\textbf{null}\rangle\varphi} \qquad \langle\textbf{null}\rangle\text{-E*}\quad \frac{\langle\textbf{null}\rangle\varphi}{\varphi}$$

$$\langle:=\rangle\text{-I}\quad \frac{\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi}{\langle\bar{x}:=\bar{\tau}\rangle\varphi} \qquad \langle:=\rangle\text{-E*}\quad \frac{\langle\bar{x}:=\bar{\tau}\rangle\varphi}{\textbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \textbf{in}\ \varphi}$$

$$\langle;\rangle\quad \langle\alpha;\beta\rangle\varphi\supset\subset\langle\alpha\rangle\langle\beta\rangle\varphi$$

$$\langle\mu\rangle\quad \langle\mu\xi.\alpha\rangle\varphi\supset\subset\langle\alpha_{\xi}[\mu\xi.\alpha]\rangle\varphi$$

$$\text{Choice-I}\quad \frac{\begin{matrix}[\pi_1] & & [\pi_n]\\ \vdots & & \vdots\\ \vee_i\pi_i\ \wedge_i\triangle\pi_i\ \langle\alpha_1\rangle\varphi & \cdots & \langle\alpha_n\rangle\varphi\end{matrix}}{\langle\pi_1\rightarrow\alpha_1|\cdots|\pi_n\rightarrow\alpha_n\rangle\varphi}$$

$$\text{Choice-E*}\quad \frac{\pi_k\ \langle\pi_1\rightarrow\alpha_1|\cdots|\pi_n\rightarrow\alpha_n\rangle\varphi}{\langle\alpha_k\rangle\varphi}\quad (\textit{for each } k)$$

$$\frac{\langle\pi_1\rightarrow\alpha_1|\cdots|\pi_n\rightarrow\alpha_n\rangle\varphi}{\wedge_i\triangle\pi_i\wedge\vee_i\pi_i}$$

where $\bar{\tau}$ are executable terms and π_i's are boolean expressions.

Just like formulas, we also impose the following condition for program variables: all the program variables having the same program variable symbol must be compatible in each proof. We can substitute programs to program variables through a proof:

Proposition 3.4. *If $\bar{\alpha}$ are substitutible for $\bar{\xi}$ and*

$$\vdash_{\text{PLi}}\Gamma\rightarrow\varphi,$$

then

$$\vdash_{\text{PLi}}\Gamma_{\bar{\xi}}[\bar{\alpha}]\rightarrow\varphi_{\bar{\xi}}[\bar{\alpha}].$$

According to $\forall$-$\langle\alpha\rangle$ rule, pure left variables satisfy the following property:

Proposition 3.5. *Let $\bar{x}$ be pure left variables of α, then*

$$\vdash_{\mathrm{PLi}} \langle\alpha\rangle\varphi \leftrightarrow \forall\bar{x}.\langle\alpha\rangle\varphi.$$

Substitution Lemma for $\mathbb{PL}$i can be stated as follows:

Proposition 3.6. (Substitution Lemma for $\mathbb{PL}$i) *Suppose φ is program-free, $\bar{\tau}$ are substitutible for $\bar{x}$ in φ and $\bar{a}$ are fresh variables compatible with $\bar{x}$, then*

(i) *if $\bar{x}$ are unmarked, then*

$$\vdash_{\mathrm{PLi}} \mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \langle\alpha\rangle\varphi \leftrightarrow \triangle\bar{\tau} \wedge \langle\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}],$$

(ii) *if $\bar{x}$ are marked, then*

$$\vdash_{\mathrm{PLi}} \bar{a} = \bar{\tau} \wedge (\langle\alpha\rangle\varphi)^{\bar{a}}_{\bar{x}} \leftrightarrow \bar{a} = \bar{\tau} \wedge \langle \mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]$$

and

$$\vdash_{\mathrm{PLi}} \mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \langle\alpha\rangle\varphi \leftrightarrow \triangle\bar{\tau} \wedge \langle \mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \alpha\rangle\varphi_{\bar{x}}[\bar{\tau}],$$

(iii) *if $\bar{x}$ are marked but $\bar{x} \cap (\mathbb{LV}\alpha \cup \mathbb{RV}\alpha) = \phi$, then*

$$\vdash_{\mathrm{PLi}} \bar{a} = \bar{\tau}, (\langle\alpha\rangle\varphi)^{\bar{a}}_{\bar{x}} \leftrightarrow \bar{a} = \bar{\tau}, \langle\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]$$

and

$$\vdash_{\mathrm{PLi}} \mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \langle\alpha\rangle\varphi \leftrightarrow \triangle\bar{\tau} \wedge \langle\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}].$$

3.2. Construction part $\mathbb{PL}\mathrm{i}_C$

In $\langle\alpha\rangle\varphi$, $\langle\alpha\rangle$ indicates how to obtain a state (of variables) satisfying φ. So $\langle\alpha\rangle$ plays like an existential quantifier. Hence, given a program α together with a verification proof $\Sigma \vdash \langle\alpha\rangle\varphi$, we may construct a proof of $\exists\bar{x}.\varphi$ in the underlying logic from Σ. This is not always the case, but we can construct such a proof for every proof in a subsystem of $\mathbb{PL}$i described below.

Now we describe a formal systems $\mathbb{PL}\mathrm{i}_C$ which is a subsystem of $\mathbb{PL}$i. $\mathbb{PL}\mathrm{i}_C$ is called a *construction part* of $\mathbb{PL}$i.

Axioms and Rules 3.7. Choice-E_C rule is similar to Choice-E except all the left variables of α_i's are equal.

Definition. $\mathbb{PL}\mathrm{i}_C$ is a formal system obtained from $\mathbb{PL}$i by omitting inference rules indicated by (*), that is, $\langle\alpha\rangle$-$\supset$*, $\langle\alpha\rangle$-$\wedge$*, $\langle\alpha\rangle$-$\forall$*, $\langle$**var**$\rangle$-E*, $\langle$**null**$\rangle$-E*, $\langle:=\rangle$-E* and Choice-E* and adding Choice-E_C.

Definition. We associate each formula φ with a formula $\varphi^\dagger$ as follows:

(i) if φ is an atomic formula, then $\varphi^\dagger$ is φ;
(ii) $(\wedge_i\varphi_i)^\dagger$ is $\wedge_i(\varphi_i^\dagger)$;
(iii) $(\vee_i\varphi_i)^\dagger$ is $\vee_i(\varphi_i^\dagger)$;
(iv) $(\varphi \supset \psi)^\dagger$ is $(\varphi^\dagger) \supset (\psi^\dagger)$;
(v) $(\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \varphi)^\dagger$ is $\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ (\varphi^\dagger)$;

(vi) $(\forall \bar{x}.\varphi)^\dagger$ is $\forall \bar{x}.(\varphi^\dagger)$;
(vii) $(\exists \bar{x}.\varphi)^\dagger$ is $\exists \bar{x}.(\varphi^\dagger)$;
(viii) if $\mathbf{LV}\alpha$ is $\bar{x}$ then $(\langle\alpha\rangle\varphi)^\dagger$ is $\exists \bar{x}.(\varphi^\dagger)$.

If Γ is $\varphi_1, \ldots, \varphi_n$, $\Gamma^\dagger$ denotes $\varphi_1^\dagger, \ldots, \varphi_n^\dagger$.

The following theorem shows that we can construct a proof in the underlying logic from every $\mathbf{PLi}_C$ proof.

Theorem 3.8. *If*

$$\vdash_{\mathbf{PLi}_C} \Gamma \to \varphi,$$

then

$$\vdash_{\mathbf{UL1}} \Gamma^\dagger \to \varphi^\dagger.$$

§4. Relation to Hoare Logic

PLi is not complete but have practically sufficient power to demonstrate verification proofs. In particular, we can simulate Hoare style verification in **PLi**. We first define a **while** statement in our framework.

4.1. While programs

Definition. We define a **while** program construct as the following abbreviation:

$$\textbf{while } \pi \textbf{ do } \alpha \triangleq \mu\xi{:}(\bar{x};\bar{y}).(\pi \to \alpha;\ \xi \mid \neg\pi \to \textbf{null})$$

where $\bar{x} = \mathbf{LV}\alpha$, $\bar{y} = \mathbf{RV}\alpha \cup \mathbf{FV}\pi$.

Proposition 4.1. *For a* **while** *program* **while** π do α, *the following holds*:

$$\vdash_{\mathbf{PLi}} \langle \textbf{while } \pi \textbf{ do } \alpha\rangle\varphi \supset \subset \langle \pi \to \alpha;\ \textbf{while } \pi \textbf{ do } \alpha \mid \neg\pi \to \textbf{null}\rangle\varphi.$$

Corollary 4.2. *For* **while**, *the following derived rules hold*:

$$\langle\textbf{while}\rangle \quad \frac{\pi \ \triangle\pi\ \langle\alpha\rangle\langle \textbf{while } \pi \textbf{ do } \alpha\rangle\varphi}{\langle \textbf{while } \pi \textbf{ do } \alpha\rangle\varphi} \qquad \frac{\neg\pi \ \triangle\pi\ \varphi}{\langle \textbf{while } \pi \textbf{ do } \alpha\rangle\varphi}.$$

Thus we can treat **while** programs in **PLi**. On the other hand, we can build *a program analysis logic for while programs*, **PLw**, by regarding **while** as a primitive program construct and $\langle$**while**$\rangle$ rules as its logical inference rules.

4.2. Total Hoare logic

Definition. Let φ and ψ be **UL1** formulas and α be a program. The following expression is called an *H-formula*:

$$\{\varphi\}\alpha\{\psi\}.$$

We now define the *total Hoare Logic* (tHL).

Definition. *Formulas* of tHL are formulas of UL1 or H-formulas.

Axioms and Rules 4.3. Axioms and inference rules of tHL are those of UL1 and the following:

$\{\mathbf{null}\}$ $\quad \{\varphi\}\,\mathbf{null}\,\{\varphi\}$

$\{:=\}$ $\quad \{\triangle\bar{\tau} \wedge \varphi_{\bar{x}}[\bar{\tau}]\}\bar{x}:=\bar{\tau}\{\varphi\}$
where $\bar{\tau}$ are executable.

$\{;\}$ $\quad \dfrac{\{\varphi\}\alpha\{\theta\} \;\; \{\theta\}\beta\{\psi\}}{\{\varphi\}\alpha;\beta\{\psi\}}$
where $\mathbb{L}\mathbb{V}\alpha$ do not occur in the assumptions of $\{\theta\}\beta\{\psi\}$.

$\{\text{Choice}\}$ $\quad \dfrac{\vee_i\pi_i \;\; \wedge_i\triangle\pi_i \;\; \{\pi_1 \wedge \varphi\}\alpha_1\{\psi\}\cdots\{\pi_n \wedge \varphi\}\alpha_n\{\psi\}}{\{\varphi\}\pi_1 \to \alpha_1 | \cdots | \pi_n \to \alpha_n\{\psi\}}$

$\mathbb{TI}(\prec, \lambda\bar{x}.\chi)$
$\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi$

$\{\mathbf{while}\}$ $\quad \triangle\pi$
$\dfrac{\{\bar{a}=\bar{\tau} \wedge \triangle\pi \wedge \pi \wedge \varphi\}\alpha\{(\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi\}}{\{\varphi\}\ \mathbf{while}\ \pi\ \mathbf{do}\ \alpha\{\varphi \wedge \neg\pi\}}$
where $\mathbb{L}\mathbb{V}\alpha$ do not occur in the assumptions above the H-formula
$\{\bar{a}=\bar{\tau} \wedge \triangle\pi \wedge \pi \wedge \varphi\}\alpha\{(\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi\}$

Conseq $\quad \dfrac{\varphi \supset \varphi' \;\; \{\varphi'\}\alpha\{\psi'\} \;\; \psi' \supset \psi}{\{\varphi\}\alpha\{\psi\}}$
where $\mathbb{L}\mathbb{V}\alpha$ do not occur in the assumptions above $\psi' \supset \psi$.

Definition. We associate each tHL formula Φ with a PLi formula $\Phi^{\S}$ as follows:

- if Φ is a UL1 formula φ, then $\Phi^{\S}$ is φ;
- if Φ is a H-formula $\{\varphi\}\alpha\{\psi\}$, then $\Phi^{\S}$ is $\varphi \supset \langle\alpha\rangle\psi$.

If Γ is $\Phi_1,\ldots,\Phi_n$, $\Gamma^{\S}$ denotes $\Phi_1^{\S},\ldots,\Phi_n^{\S}$.

The following theorem shows that tHL is contained in $\mathbb{PL}\mathrm{i}_{\mathrm{C}}$.

Theorem 4.4. *If*

$$\vdash_{\mathrm{tHL}} \Gamma \to \Phi,$$

then

$$\vdash_{\mathrm{PLi_C}} \Gamma^{\S} \to \Phi^{\S}.$$

Theorem 4.4 together with Theorem 3.8 yields:

Corollary 4.5. *If Γ is a set of* **UL1** *formulas and*

$$\vdash_{\text{tHL}} \Gamma \to \{\varphi\}\alpha\{\psi\},$$

then

$$\vdash_{\text{UL1}} \Gamma \to \varphi \supset \exists \bar{x}\psi$$

where $\bar{x} = \mathbf{LV}\alpha$.

§5. Program Extraction

In this section, we shall state a method to extract a program from a proof of its specification.

For the simplicity, we shall restrict occurrences of existential quantifiers which bind marked variables, that is, occurrences of existential quantifiers which are supposed to correspond to programs.

5.1. Underlying logic $\tilde{\exists}$-UL1

We now define an auxiliary formal system $\tilde{\exists}$-**UL1**. In $\tilde{\exists}$-**UL1**, occurrences of variables bound by existential quantifiers are restricted. In $\exists \bar{x}.\varphi$, all the variables in $\bar{x}$ are entirely marked or all are entirely unmarked. We write $\tilde{\exists}\bar{x}.\varphi$ when $\bar{x}$ are marked variables.

Definition. *Formulas* of $\tilde{\exists}$-**UL1** are inductively defined as those of **UL1** but the following exceptions:

- if π is a boolean expression, $\Delta\pi$ is a formula;
- if φ is a formula and $\bar{x}$ are distinct unmarked variables, then $\exists \bar{x}.\varphi$ is a formula;
- if φ is a formula and $\bar{x}$ are distinct marked variables, then $\tilde{\exists}\bar{x}.\varphi$ is a formula.

Axioms and Rules 5.1. We need axioms for $\Delta\pi$: that is the Boolean axioms for **PLi**. Inference rules for a $\tilde{\exists}$ quantifier are:

$$\tilde{\exists}\text{-I}\ \frac{\textbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \textbf{in}\ \varphi}{\tilde{\exists}\bar{x}.\varphi} \qquad \tilde{\exists}\text{-E}\ \frac{\tilde{\exists}\bar{x}.\varphi \qquad \begin{array}{c}[\varphi^{\bar{a}}_{\bar{x}}] \\ \vdots \\ \psi\end{array}}{\psi}\langle \bar{a} \rangle$$

where $\bar{\tau}$ are executable and $\bar{a}$ are marked.

$\vee$-E rule is also modified as follows:

- If its minor premise ψ contains positive occurrences of $\tilde{\exists}$ quantifiers,

then the major premise must be a boolean expression and the rule is of the following form:

$$\frac{\bigvee_i \pi_i \quad \begin{array}{c}[\pi_1]\\ \vdots\\ \psi\end{array} \cdots \begin{array}{c}[\pi_n]\\ \vdots\\ \psi\end{array}}{\psi}$$

- Otherwise, $\vee$-E is the same as in UL1.

5.2. Program Extraction Theorem

Definition. An *SP-context* and an *SN-context* are defined as follows:

- If φ is a $*$-free formula, then φ is an SP-context and SN-context;
- $*_i$ is an SP-context;
- If $\mathscr{A}_1, \ldots, \mathscr{A}_n$ are SP-context (SN-context), then so are $\wedge_i \mathscr{A}_i$ and $\vee_i \mathscr{A}_i$;
- If $\mathscr{A}$ is an SP-context (SN-context), then so are $\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \mathscr{A}$, $\forall \bar{x}.\mathscr{A}$, $\exists \bar{x}.\mathscr{A}$ and $\tilde{\exists} \bar{x}.\mathscr{A}$;
- If φ is a $*$-free formula and $\mathscr{A}$ is an SP-contexts, then $\varphi \supset \mathscr{A}$ is an SP-context;
- If $\mathscr{A}$ is an SP-context and $\mathscr{B}$ is an SN-context, then $\mathscr{A} \supset \mathscr{B}$ is an SN-context.

An occurrence of subformula expressed by an SP-context is said to be an *SP occurrence* or simply a *positive occurrence*. An *SN* or *negative occurrence* is similarly defined.

Definition. An SP-context (SN-context) $\mathscr{A}[\bar{*}]$ is said to be an *SSP-context* (resp. *SSN-context*) if each $*_i$ does not occur in any scope of $\vee$ or $\exists$. An occurrence represented by an SP (resp. SN, SSP, SSN)-context is said to be *SP* (resp. *SN*, *SSP*, *SSN*). An *SNP-context* (resp. *SSNP-context*) is a context in which each occurrence of $*_i$'s is SP or SN (resp. SSP or SSN).

Definition. An occurrence of subformula expressed by an SP-context is said to be an *SP occurrence* or simply a *positive occurrence*. An *SN* or *negative occurrence* is similarly defined.

An SP-context (SN-context) $\mathscr{A}[\bar{*}]$ is said to be an *SSP-context* (resp. *SSN-context*) if each $*_i$ does not occur in any scope of $\vee$ or $\exists$. An occurrence represented by an SP (resp. SN, SSP, SSN)-context is said to be *SP* (resp. *SN*, *SSP*, *SSN*). An *SNP-context* (resp. *SSNP-context*) is a context in which each occurrence of $*_i$'s is SP or SN (resp. SSP or SSN).

Definition. A $\tilde{\exists}$-*formula* is a formula which contains $\tilde{\exists}$ quantifiers. Especially for a $\tilde{\exists}$-free formula φ, $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\varphi$ is said to be $\tilde{\exists}$-*prefixed formula* and $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n$ is called the $\tilde{\exists}$-*prefix* of the formula.

A $\tilde{\exists}$-*SP formula* is a formula obtained from a $\tilde{\exists}$-free SP-context $\mathscr{A}[\bar{*}]$ by substituting $\tilde{\exists}$-prefixed formulas to $*_i$'s. A $\tilde{\exists}$-*SN*, $\tilde{\exists}$-*SSP* or $\tilde{\exists}$-*SNP formula* is similarly defined.

A $\tilde{\exists}$-*SNP sequent* is a sequent $\Gamma \to \varphi$ such that all the formula in Γ are $\tilde{\exists}$-SP and φ is $\tilde{\exists}$-SNP. A $\tilde{\exists}$-*SNP proof* Σ is a proof such that all the sequents for the subproofs of Σ are $\tilde{\exists}$-SNP.

We weaken **CV** for $\tilde{\exists}$-SNP formulas, sequents and proofs.

Definition. A $\tilde{\exists}$-SNP formula satisfies $\tilde{\exists}$-*condition for variables* ($\tilde{\exists}$-**CV**) if

- No marked variables occur both free and bound.
- For every $\tilde{\exists}$-prefix $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n$ occurring in it, $\bar{x}_i$'s are mutually disjoint.
- It is obtained from an SNP context $\mathscr{A}[\bar{*}]$ enjoying (strong) **CV** for marked variables and none of the variables occurring in the $\tilde{\exists}$-prefixes are bound in $\mathscr{A}[\bar{*}]$.

A $\tilde{\exists}$-SNP sequent or proof satisfies $\tilde{\exists}$-*condition for variables* ($\tilde{\exists}$-**CV**) if

- No marked variables occur both free and bound.
- All the formulas occurring in it satisfy $\tilde{\exists}$-**CV**.
- In a proof, all the marked eigen variables are distinct.

For example, $\forall y.(F(y) \supset \tilde{\exists}x.(G(x, y) \wedge \forall y.H(x, y)))$ enjoys $\tilde{\exists}$-**CV** even if y is a marked variable, but $\forall x.(F(x, y) \supset (\tilde{\exists}x.G(x, y)))$ does not.

If a proof Σ of $\Gamma \to \varphi$ does not satisfy $\tilde{\exists}$-**CV**, then by Proposition 2.11 we can construct a proof Σ' of a variant of $\Gamma \to \varphi$ such that Σ' satisfies **CV** (hence satisfies $\tilde{\exists}$-**CV** because a proof satisfying **CV** also satisfies $\tilde{\exists}$-**CV**). We can therefore consider only the proofs satisfying $\tilde{\exists}$-**CV** without loss of generality.

Definition. For each occurrence of $\tilde{\exists}$-formula in a $\tilde{\exists}$-SNP formula, we define the *p-type* of the occurrence.

- Let $\mathscr{A}[*]$ be an SNP-context expressing the occurrence of $\tilde{\exists}$-prefixed formula $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\varphi$, $\bar{y}$ be all the marked variables which are bound in the context, $\bar{z}$ be all the marked variables which occur free in φ but do not belong to $\bar{x}$. Then the *p-type* of the occurrence is $(\bar{x}_1, \ldots, \bar{x}_n; \bar{y} \cup \bar{z})$.

We next define the p-type of each occurrence of $\tilde{\exists}$-formula in a $\tilde{\exists}$-SNP sequent $\Gamma \to \varphi$.

- For an SP occurrence of $\tilde{\exists}$-formula in the conclusion φ, the p-type is that of the corresponding occurrence in the universal closure of $\wedge\Gamma \supset \varphi$.
- For an SN occurrence of $\tilde{\exists}$-formula in the conclusion φ, the p-type is that in the formula φ.
- For an SP occurrence of $\tilde{\exists}$-formula in a formula ψ in Γ, the p-type is that in the formula ψ

Example 3. Consider the following sequents.

(i) $\rightarrow \forall n \tilde{\exists} m.F(n, m)$ (n, m: marked);
(ii) $a \neq 0,\ \forall x.(x < a \supset \tilde{\exists} z.F(x, z)) \rightarrow \tilde{\exists} z.F(a-1, z)$ (a, n, z: marked);
(iii) $\forall x.(x < a \supset \tilde{\exists} z.F(x, z)) \rightarrow \forall x.(x < a \supset \tilde{\exists} z.F(x, z))$ (a, n, z: marked);
(iv) $a \neq 0,\ F(a-1, z) \rightarrow \tilde{\exists} z.F(a, z)$ (a, z: marked);
(v) $\rightarrow \exists z.(\forall x.(\varphi[x] \supset \tilde{\exists} y.\psi[x, y, z]))$ (z: unmarked, x, y: marked);
(vi) $\rightarrow \forall f.(\mathit{formula}(f) \supset (\exists q.\mathit{proof}(q, f) \supset \tilde{\exists} p.\mathit{proof}(p, f)))$
(q: unmarked, f, p, q: marked).

The p-types of $\tilde{\exists}$-prefixed formulas in these sequences are:

(i) the p-type of $\tilde{\exists} m.F(n, m)$ is $(m; n)$;
(ii) the p-type of $\tilde{\exists} z.F(x, z)$ is $(z; x)$ and the p-type of $\tilde{\exists} z.F(a-1, z)$ is $(z; a)$;
(iii) the p-type of $\tilde{\exists} z.F(x, z)$ in the left hand side is $(z; x)$ and the p-type of $\tilde{\exists} z.F(x, z)$ in the right hand side is $(z; x, a)$;
(iv) the p-type of $\tilde{\exists} z.F(x, z)$ is $(z; a, z)$;
(v) the p-type of $\tilde{\exists} y.\psi[x, y, z]$ is $(y; x)$;
(vi) the p-type of $\tilde{\exists} p$: $\mathit{proof}(p, f)$ is $(p; f)$.

Now we are in the position to state our main theorem.

Theorem 5.2. (Program Extraction Theorem) *Let $\Gamma[*]$ be a set of $\tilde{\exists}$-SP contexts and $\mathscr{A}[*]$ be a $\tilde{\exists}$-SP context. Suppose*

$$(1) \qquad \sum \vdash_{\tilde{\exists}\text{-ULI}} \Gamma[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\varphi}] \rightarrow \mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}],$$

and Σ satisfies the following conditions:

(i) $\tilde{\exists}$-CV *holds for* Σ;
(ii) *Every formula occurring in* Σ *is* $\tilde{\exists}$-*SP*;
(iii) *Every minor premise in* $\vee$-E *or* $\tilde{\exists}$-*E is* $\tilde{\exists}$-*SSP*.

Then for distinct program variables $\bar{\xi}$ corresponding to the occurrences of $\tilde{\exists}$-prefixed formulas in $\Gamma[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\varphi}]$, there exist a proof Σ' and programs $\bar{\alpha}$ such that

$$\sum' \vdash_{\text{PLi}} \Gamma[\overline{\langle\xi\rangle\varphi}] \rightarrow \mathscr{A}[\overline{\langle\alpha\rangle\psi}],$$

where each of $\bar{\xi}$ and $\bar{\alpha}$ has the p-type of its corresponding $\tilde{\exists}$-prefixed formula in the sequent, and all the program variables occurring in each α_i belong to $\bar{\xi}$.

Moreover, this theorem holds even if we augment the right variables of ξ_i's as follows:

Let $\mathscr{B}[\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\varphi]$ be the formula corresponding to ξ, then we may add arbitrary number of free marked variables occurring in $\mathscr{B}$ to the right variables of ξ.

Condition (ii) of the theorem implies that each instance of Axiom EM, $\varphi \vee \neg \varphi$, has no $\tilde{\exists}$ quantifiers. Hence Σ is considered as a constructive proof with respect to $\tilde{\exists}$. We can omit the condition (iii) of this theorem and shall discuss it in Section 5.4.

In the rest of this paper, every context is assumed to be an SNP-context, unless we explicitly mention otherwise.

5.3. Proof of the Program Extraction Theorem

We prove the theorem by induction on the length of proofs. We only demonstrate typical cases. Other cases are similar or straightforward. In this section, $\bar{y}$ stands for $\bar{y}_1, \ldots, \bar{y}_m$ and $\bar{z}$ stands for $\bar{z}_1, \ldots, \bar{z}_n$.

In what follows, we shall construct a proof Σ' from (1) such that it satisfies the following additional property:

(A) if a is a free variable occurring in Σ which is not an eigen variable in Σ, then a does not occur as a bound variable nor an eigen variable in Σ'.

Since we replace the part of the form $\tilde{\exists}\bar{x}.\varphi$ by $\langle\alpha\rangle\varphi$ for some program α, $\bar{x}$ in the latter become free though they are bound in the former. But $\tilde{\exists}$-**CV** guarantees that $\bar{x}$ are pure left variables. By Proposition 3.5 they can be bound after transformation. Hence the eigen variable condition of the obtained proof is guaranteed and renaming variables (occurring in $\forall$-I rule and so forth) causes no problem.

Lemma 5.3. *We can add any new right variables to a program without changing its meaning. That is, for programs $\bar{\alpha}$ and given marked variables $\bar{a}$, we can find programs $\bar{\alpha}'$ whose right variables are $\bar{a}$ and* **RV**α, *such that*

$$\vdash_{\mathrm{PLi}} \mathscr{A}[\overline{\langle\alpha\rangle\psi}] \leftrightarrow \mathscr{A}[\overline{\langle\alpha'\rangle\psi}].$$

Proof. Let $(\bar{u}; \bar{v})$ be the p-type of α and $\bar{a}$ be the added right variables. Take **null**:$(\bar{u}; \bar{v}, \bar{a}); \alpha$ as α'.

Lemma 5.4. *Suppose*

$$\sum \vdash_{\mathrm{PLi}} \Gamma[\overline{\langle\xi\rangle\varphi}] \rightarrow \mathscr{A}[\overline{\langle\alpha\rangle\psi}]$$

and $\bar{a}$ are marked variables which do not occur as left variables, bound variables nor as eigen variables in Σ. Let $\bar{\xi}'$ be program variables which is obtained by adding right variables $\bar{a}$ to $\bar{\xi}$, α' (resp. Σ') be a program (resp. a proof) obtained from α (resp. Σ) by adding $\bar{a}$ as right variables to each $\bar{\xi}$ occurring in it. Then

$$\sum' \vdash_{\mathrm{PLi}} \Gamma[\overline{\langle\xi'\rangle\varphi}] \rightarrow \mathscr{A}[\overline{\langle\alpha'\rangle\psi}],$$

Lemma 5.5. *Suppose that $\mathscr{A}[*]$ is an SSP context and*

$$\vdash_{\mathrm{PLi}} \Gamma \to \mathscr{A}[\langle\alpha\rangle\varphi].$$

Then we can reduce superfluous right variables from the p-type of α. *That is, if* $x \in \mathbb{RV}\alpha$, x *does not occur free in* Γ, φ *nor* $\mathscr{A}[*]$, *and* x *does not bound at* $\mathscr{A}[*]$, *then* x *can be omitted from the right variables of* α *as follows*:

(i) *If* $x \notin \mathbb{LV}\alpha$, *then*

$$\vdash_{\mathrm{PLi}} \Gamma \to \mathscr{A}[\langle \textbf{local } x \textbf{ in } \alpha\rangle\varphi].$$

(ii) *If* $x \in \mathbb{LV}\alpha$, *then*

$$\vdash_{\mathrm{PLi}} \Gamma \to \mathscr{A}[\langle \textbf{local } x \textbf{ in } \alpha\rangle\varphi].$$

In this case x *can occur free in* **var** φ.

5.3.1. Base case If the proof solely consists of an assumption, the assumption and the conclusion are the same formula, say $\mathscr{A}[\,\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m \cdot \varphi}\,]$. We must find α_i's such that

$$\vdash_{\mathrm{PLi}} \mathscr{A}[\,\overline{\langle\xi\rangle\varphi}\,] \to \mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,].$$

For the p-types of ξ_i and α_i, the following must hold:

$$\mathbb{LV}\xi_i = \mathbb{LV}\alpha_i \quad \text{and} \quad \mathbb{RV}\xi_i \subseteq \mathbb{RV}\alpha_i.$$

Obviously,

$$\vdash_{\mathrm{PLi}} \mathscr{A}[\,\overline{\langle\xi\rangle\varphi}\,] \to \mathscr{A}[\,\overline{\langle\xi\rangle\varphi}\,].$$

If the right variables of α_i are the same as those of the ξ_i, then we can take ξ_i as α_i. If there are right variables of α_i which are not those of ξ_i, then we can apply Lemma 5.3 to obtain α_i.

Now we consider each inference rule. If either a formula appearing as the conclusion or all the discharged formulas have no $\bar{\exists}$-quantifier, this case is straightforward from its induction hypothesis. Therefore we only consider the cases that the conclusion or at least one of discharged formulas contains some $\bar{\exists}$-quantifiers.

5.3.2. $\vee$-I Consider the following simple case:

$$\frac{\begin{array}{c}\Sigma \\ \mathscr{A}[\,\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m \cdot \varphi}\,]\end{array}}{\mathscr{A}[\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m \cdot \varphi}] \vee \mathscr{B}[\,\overline{\exists \bar{z}_1 \ldots \exists \bar{z}_n \cdot \psi}\,]}.$$

From the induction hypothesis, we have

$$\frac{\Sigma'}{\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,]}.$$

Then we obtain the following proof:

$$\frac{\dfrac{\Sigma'}{\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,]}}{\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,] \vee \mathscr{B}[\overline{\langle\mathbf{null}\rangle\psi}]}\ \vee\text{-I}$$

where **null**'s have appropriate p-types.

5.3.3. $\vee$**-E** Consider the following proof:

$$\frac{\prod_0 \quad \prod_1 \qquad \overset{[\pi_1]}{\Sigma_1}}{\vee_i \pi_i \quad \triangle\,\pi_1 \ldots \quad \mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\,] \ldots}$$
$$\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\,].$$

From the induction hypothesis we have the following proofs:

$$\prod'_0 \quad \prod'_1 \qquad \overset{\pi_1}{\Sigma'_1}$$
$$\vee_i \pi_i, \quad \triangle\,\pi_1 \ldots \quad \text{and} \quad \mathscr{A}[\overline{\langle\alpha_1\rangle\varphi}] \ldots.$$

Lemma 5.6. *If $\mathscr{A}[*]$ is an SSP context, none of the free variables occurring in π_i's is bound in the context $\mathscr{A}[*]$, then the following derived rule holds:*

$$\frac{\begin{array}{c}[\pi_1]\\ \vdots\end{array}}{\vee_i \pi_i \quad \triangle\,\pi_i \ldots \quad \mathscr{A}[\overline{\langle\alpha_1\rangle\varphi}] \ldots}$$
$$\mathscr{A}[\overline{\langle\pi_1 \to \alpha_1 | \ldots | \pi_n \to \alpha_n\rangle\varphi}].$$

$\tilde{\exists}$**-CV** guarantees the conditions of this lemma. Since π_i is discharged, the p-type of each $\tilde{\exists}$-formula in the conclusion may have less right variables of the corresponding p-type in the premise. In this case we can decrease the right variables of α_i by Lemma 5.5. Thus this lemma establishes the $\vee$-E case.

5.3.4. let-E We only consider the case where all the variables bound by the **let** quantifier are marked. Moreover we assume that both the major and the minor

premises are $\bar{\exists}$ formulas. Now the proof to be considered is of the following form:

$$\dfrac{\begin{array}{cc}\prod & \sum \\ \textbf{let } \bar{x} \leftarrow \bar{\tau} \textbf{ in } \mathscr{A}[\,\overline{\bar{\exists}\bar{y}_1 \ldots \bar{\exists}\bar{y}_m.\varphi}\,] & \mathscr{B}[\,\overline{\bar{\exists}\bar{z}_1 \ldots \bar{\exists}\bar{z}_n.\psi}\,]\end{array} \qquad \begin{array}{c}[\bar{a} = \bar{\tau},\ \mathscr{A}[\,\overline{\bar{\exists}\bar{y}_1 \ldots \bar{\exists}\bar{y}_m.\varphi}\,]^{\bar{a}}_{\bar{x}}] \\ \vdots \\ \mathscr{B}[\,\overline{\bar{\exists}\bar{z}_1 \ldots \bar{\exists}\bar{z}_n.\psi}\,]\end{array}}{\mathscr{B}[\,\overline{\bar{\exists}\bar{z}_1 \ldots \bar{\exists}\bar{z}_n.\psi}\,]}\langle \bar{a} \rangle$$

From the induction hypothesis, we have

$$\begin{array}{cc}\prod' & \sum' \\ \textbf{let } \bar{x} \leftarrow \bar{\tau} \textbf{ in } \mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,] & \mathscr{B}[\,\overline{\langle\beta\rangle\psi}\,]\end{array} \quad \text{and} \quad \begin{array}{c}\bar{a} = \bar{\tau},\ \mathscr{A}[\,\overline{\langle\xi\rangle\varphi}\,]^{\bar{a}}_{\bar{x}} \\ \vdots \\ \mathscr{B}[\,\overline{\langle\beta\rangle\psi}\,]\end{array}.$$

Here α_i's and the corresponding ξ_i's may have different p-types. But this discrepancy comes from the discrepancy of free variables occurring assumptions above the major premise. In view of (A) and by Lemma 5.4, we can however regard α_i's and ξ_i's have the same p-type.

Since $\bar{a}$ are marked variables, eigen variables $\bar{a}$ may occur in $\bar{\beta}$, in the right hand side proof Σ. Hence we must bind $\bar{a}$ in β before applying **let**-E rule to these two subproofs. The following lemma shows how $\bar{a}$ can be bound in β.

Lemma 5.7. *If $\bar{x}$ are marked and not bound in $\mathscr{A}[*]$, and $\bar{\tau}$ are substitutible in a context $\mathscr{A}[*]$, then the following derived rule holds*:

$$\dfrac{\begin{array}{c}[\bar{a} = \bar{\tau}] \\ \vdots \\ \triangle\bar{\tau} \quad \mathscr{A}[\,\overline{\langle\alpha^{\bar{a}}_{\bar{x}}\rangle\varphi}\,]\end{array}}{\mathscr{A}[\,\overline{\langle \textbf{let } \bar{x} \leftarrow \bar{\tau} \textbf{ in } \alpha\rangle\varphi}\,]}\langle \bar{a} \rangle$$

From this lemma and by paying attention to names of variables, we have the following proof:

$$\dfrac{\dfrac{\prod'}{\textbf{let } \bar{x}\leftarrow\bar{\tau} \textbf{ in } \mathscr{A}[\overline{\langle\alpha\rangle\varphi}]} \quad \dfrac{\dfrac{\dfrac{\dfrac{\overset{(1)}{\bar{a}=\bar{\tau}},\ \dfrac{\overset{(1)}{\bar{a}=\bar{\tau}} \quad \overset{\text{(let-E)}}{\bar{b}=\bar{\tau}}}{\bar{a}=\bar{b}} \quad \dfrac{\mathscr{A}[\overline{\langle\alpha\rangle\varphi}]^{\bar{b}}_{\bar{x}}}{\mathscr{A}[\overline{<\alpha>\tau}]^{\bar{a}}_{\bar{x}}}\ \text{Rename}}{\sum'_{\bar{\xi}}[\bar{\alpha}]} \quad \overset{\text{(let-E)}}{\bar{b}=\bar{\tau}}}{\mathscr{B}[\overline{\langle\beta_{\bar{\xi}}[\bar{\alpha}]\rangle\psi}]} \quad \dfrac{\mathscr{B}[\overline{\langle\beta_{\bar{\xi}}[\bar{\alpha}]\rangle\psi}]}{\mathscr{B}[\overline{\langle((\beta^{\bar{a}}_{\bar{a}})_{\bar{\xi}}[\bar{\alpha}])^{\bar{a}}_{\bar{u}}\rangle\psi}]}\equiv(*)}{\mathscr{B}[\overline{\langle\textbf{let } \bar{u}\leftarrow\bar{\tau} \textbf{ in } (\beta^{\bar{a}}_{\bar{a}})_{\bar{\xi}}[\bar{\alpha}]\rangle\psi}]}\ \langle\bar{a}\rangle\ (1)\ \text{Lemma 5.7}(**)}{\mathscr{B}[\overline{\langle\textbf{let } \bar{u}\leftarrow\bar{\tau} \textbf{ in } (\beta^{\bar{a}}_{\bar{a}})_{\bar{\xi}}[\bar{\alpha}]\rangle\psi}]}\ \langle\bar{b}\rangle\text{let-E}$$

where $\bar{b}$ are fresh variables; u_i is as follows:

$$u_i \equiv \begin{cases} a_i & \text{if } x_i \in \bar{z} \\ x_i & \text{otherwise.} \end{cases}$$

(*): We take $\bar{u}$ such that $\bar{u}$ do not conflict with $\bar{z}$. Hence these two formulas are identical.

(**): From $\tilde{\exists}$-CV, $\bar{\tau}$ is substitutible in $\mathscr{B}[*]$. Hence Lemma 5.7 can be applied.

Just like $\vee$-E rule, some extra right variables may occur in $\bar{\beta}$, because $\bar{a}=\bar{\tau}$ is discharged. These extra right variables are deleted from the p-types of the finally obtained program by Lemma 5.5.

5.3.5. $\exists$-E The proof to be considered is of the following form:

$$\dfrac{\dfrac{\prod}{\exists\bar{x}.\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}]} \quad \dfrac{\sum}{\mathscr{B}[\overline{\tilde{\exists}\bar{z}_1\ldots\tilde{\exists}\bar{z}_n.\psi}]}}{\mathscr{B}[\overline{\tilde{\exists}\bar{z}_1\ldots\tilde{\exists}\bar{z}_n.\psi}]}\ \langle\bar{a}\rangle \qquad [\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}]^{\bar{a}}_{\bar{x}}]$$

From the induction hypothesis, we have

$$(2) \qquad \dfrac{\prod'}{\exists\bar{x}.\mathscr{A}[\overline{\langle\alpha\rangle\varphi}]} \quad \text{and} \quad \dfrac{\mathscr{A}[\overline{\langle\xi\rangle\varphi}]^{\bar{a}}_{\bar{x}}}{\sum'_{\mathscr{B}[\overline{\langle\beta\rangle\psi}]}}.$$

Using Lemma 5.4, we can assume both $\bar{\alpha}$ and $\bar{\xi}$ have the same p-types. Since $\bar{a}$ are unmarked, they do not occur in β_i's, hence also in $\mathscr{B}[\overline{\langle\beta\rangle\psi}]$. So we can simply apply $\exists$-E to (2) and obtain:

$$\dfrac{\begin{array}{cc} & A[\overline{\langle\alpha\rangle\varphi}]^{\bar{a}}_{\bar{x}} \\ \prod' & \sum'_{\xi}[\bar{\alpha}] \\ \exists\bar{x}.\mathscr{A}[\overline{\langle\alpha\rangle\varphi}] & \mathscr{B}[\overline{\langle\beta_{\xi}[\bar{\alpha}]\rangle\psi}] \end{array}}{\mathscr{B}[\overline{\langle\beta_{\xi}[\bar{\alpha}]\rangle\psi}]}\ \langle\bar{a}\rangle\exists\mathrm{E}$$

5.3.6. $\bar{\exists}$-I In general, the proof to be considered is of the following form:

$$\dfrac{\sum \quad \mathbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \mathbf{in}\ \bar{\exists}\bar{y}_1\ldots\bar{\exists}\bar{y}_m.\varphi}{\bar{\exists}\bar{x}\bar{\exists}\bar{y}_1\ldots\bar{\exists}\bar{y}_m.\varphi}$$

where $\bar{\tau}$ are executable. From the induction hypothesis, we have

$$\sum' \quad \mathbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \mathbf{in}\ \langle\alpha\rangle\varphi.$$

In α, $\bar{x}$ occur as right variables.

Lemma 5.8. *The following derived rule holds*:

$$\dfrac{\mathbf{let}\ \bar{x}\leftarrow\bar{\tau}\ \mathbf{in}\ \langle\alpha\rangle\varphi}{\langle\bar{x}:=\bar{\tau};\ \alpha\rangle\varphi.}$$

We note that even if $\bar{x}$ may appear as right variables in α, $\bar{x}$ are (pure) left variables in $\bar{x}:=\bar{\tau};\ \alpha$.

From this lemma, this case is established.

5.3.7. $\bar{\exists}$-E This case is slightly complicated. We assume that both the major and the minor premises are $\bar{\exists}$-prefixed formulas. Other cases are easily established. Now the proof to be considered is of the following form:

(3)
$$\dfrac{\begin{array}{cc} & [\bar{\exists}\bar{y}_1\ldots\bar{\exists}\bar{y}_m.\varphi^{\bar{a}}_{\bar{x}}] \\ \prod & \sum \\ \bar{\exists}\bar{x}\bar{\exists}\bar{y}_1\ldots\bar{\exists}\bar{y}_m.\varphi & \mathscr{A}[\overline{\bar{\exists}\bar{z}_1\ldots\bar{\exists}\bar{z}_n.\psi}] \end{array}}{\mathscr{A}[\overline{\bar{\exists}\bar{z}_1\ldots\bar{\exists}\bar{z}_n.\psi}]}\ \langle\bar{a}\rangle$$

From the induction hypothesis, we have

(4)
$$\prod' \quad \langle\alpha\rangle\varphi \qquad \text{and} \qquad \sum' \quad \mathscr{A}[\overline{\langle\beta\rangle\psi}].$$

By Lemma 5.4, we can augment $\mathbb{RV}\xi$ in Σ by fresh variables $\bar{b}$ compatible

with $\bar{y}$.

We now state several lemmas used in transformation of $\tilde{\exists}$-E rule.

Lemma 5.9. *If $\bar{x}$ are marked,*

$$\vdash_{\mathrm{PLi}} \varphi^{\bar{u}}_{\bar{x}} \to \langle \bar{x} := \bar{u} \rangle \varphi.$$

Lemma 5.10. *We can rename left variables in programs: if $\mathscr{A}[*]$ is a context, $\bar{x} \subseteq \mathbf{LV}\alpha$, $\bar{x}' \subseteq \mathbf{pLV}\alpha$, $\bar{x} \cap \bar{x}' = \phi$ and $\bar{x}$, $\bar{x}'$ are renamable to $\bar{u}$, $\bar{u}'$ in $\langle \alpha \rangle \varphi$, then*

$$\vdash_{\mathrm{PLi}} \mathscr{A}[\langle \alpha \rangle \varphi] \leftrightarrow \mathscr{A}[\langle \bar{u} := \bar{x};\ \alpha^{\bar{u},\bar{u}'}_{\bar{x},\bar{x}'} \rangle \varphi^{\bar{u},\bar{u}'}_{\bar{x},\bar{x}'}].$$

$\langle \alpha \rangle$-rule is generalized as follows:

Lemma 5.11. *If $\mathscr{A}[*]$ is an SSP context and $\bar{x} = \mathbf{LV}\alpha$, then the following derived rule holds:*

$$\frac{\begin{array}{c}[\varphi^{\bar{a}}_{\bar{x}}] \\ \vdots \\ \langle \alpha \rangle \varphi \quad \mathscr{A}[\overline{\psi^{\bar{a}}_{\bar{x}}}]\end{array}}{\mathscr{A}[\overline{\langle \alpha \rangle \psi}]}\ \langle \bar{a} \rangle$$

providing that

- ⓐ $\mathbf{RV}\alpha$ *are not bound in* $\mathscr{A}[*]$;
- ⓑ $\bar{a}$ *satisfy the condition like the eigen variable condition for* $\langle \alpha \rangle$*-rule; that is,* $\bar{a}$ *do not occur in* $\mathscr{A}[*]$ *nor the assumptions of* $\mathscr{A}[\overline{\psi^{\bar{a}}_{\bar{x}}}]$ *except* $\varphi^{\bar{a}}_{\bar{x}}$.

Lemma 5.12. *We can hide left variables of a program which are not referred after its execution. That is, if $\mathscr{A}[*]$ is a context, $\bar{x} \subseteq \mathbf{LV}\alpha$, $\bar{y} \subseteq \mathbf{pLV}\alpha$ and $(\bar{x} \cup \bar{y}) \cap \mathbf{FV}\varphi = \phi$, then*

$$\vdash_{\mathrm{PLi}} \mathscr{A}[\langle \alpha \rangle \varphi] \leftrightarrow \mathscr{A}[\langle \textbf{let } \bar{x} \leftarrow \bar{x} \textbf{ in local } \bar{y} \textbf{ in } \alpha \rangle \varphi].$$

Let γ be $\mathbf{null}(;\bar{v});\ \bar{y} := \bar{b}$, where $\bar{v} = \mathbf{RV}\xi$. Then γ has the same p-type as ξ. We also define $\bar{u}$ such that

$$u_i \equiv \begin{cases} a_i & \text{if } x_i \in \bar{z} \\ x_i & \text{otherwise.} \end{cases}$$

Then $\bar{u}$ are renamable to $\bar{a}$ in β; so

$$\beta_\xi[\gamma] \equiv ((\beta^{\bar{v}}_{\bar{a}})_\xi[\gamma])^{\bar{a},\bar{b}}_{\bar{u},\bar{b}}.$$

By the above lemmas and (4), we can construct the following proof:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(1)}{(\varphi^{\bar{u},\bar{b}}_{\bar{x},\bar{y}})^{\bar{a},\bar{b}}_{\bar{u},\bar{a}} \equiv \varphi^{\bar{a},\bar{b}}_{\bar{x},\bar{y}}}}{(\langle\gamma\rangle\varphi)^{\bar{a}}_{\bar{x}}}\ \text{Lemma 5.9 and}\langle\text{null}\rangle \qquad \sum'_{\xi}[\gamma]}{\langle\alpha\rangle\varphi} \qquad \mathscr{A}[\ \overline{\langle\beta_{\xi}[\gamma]\rangle\psi}\ ]}{\langle\alpha^{\bar{u},\bar{b}}_{\bar{x},\bar{y}}\rangle\varphi^{\bar{u},\bar{b}}_{\bar{x},\bar{y}}}\ \text{Lemma 5.10} \qquad \mathscr{A}[\ \overline{\langle((\beta^{\bar{u}}_{\bar{a}})_{\xi}[\gamma])^{\bar{a},\bar{b}}_{\bar{u},\bar{b}}\rangle\psi}\ ]}{\mathscr{A}[\ \overline{\langle\alpha^{\bar{u},\bar{b}}_{\bar{x},\bar{y}}\rangle\langle(\beta^{\bar{u}}_{\bar{a}})_{\xi}[\gamma]\rangle\psi}\ ]}\ \langle\bar{a},\bar{b}\rangle(1)\text{Lemma 5.11} \qquad (5)
$$

$$
\dfrac{\mathscr{A}[\ \overline{\langle(\alpha^{\bar{u},\bar{b}}_{\bar{x},\bar{y}};(\beta^{\bar{u}}_{\bar{a}})_{\xi}[\gamma])\rangle\psi}\ ]}{\mathscr{A}[\ \overline{\langle\textbf{local}\ \bar{u},\ \bar{b}\ \textbf{in}\ (\alpha^{\bar{u},\bar{b}}_{\bar{x},\bar{y}};\ (\beta^{\bar{u}}_{\bar{a}})_{\xi}[\gamma])\rangle\psi}\ ]}\ \text{Lemma 5.12}
$$

The extra right variables may occur in the finally obtained program due to the discharged assumption, but they can be deleted by Lemma 5.5.

5.3.8. $\mathbb{IND}(\prec, \lambda\bar{x}.\chi)$ Consider the following proof:

$$
\dfrac{\begin{array}{cc}[\chi^{\bar{a}}_{\bar{x}},\ \forall\bar{x}.(\chi \wedge \bar{x} \prec \bar{a} \supset \mathscr{A}[\ \overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\ ])] & \\ \prod & \sum \\ \mathbb{TI}(\prec, \lambda\bar{x}.\chi) & \mathscr{A}[\ \overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\ ]^{\bar{a}}_{\bar{x}}\end{array}}{\forall\bar{x}.(\chi \supset \mathscr{A}[\ \overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\ ]).}\ \langle\bar{a}\rangle
$$

Here we may assume the p-type of each $\tilde{\exists}$-formula in the premise agrees with the corresponding p-type in the induction hypothesis except $\bar{x}$ and $\bar{a}$ which are renamed each other. The reason is as follows: If there is discrepancy between them, it comes from marked free variables, say $\bar{b}$, occurring in the assumptions; $\bar{b}$ occur as extra right variables in the premise. In this case, it is sufficient replacing $\mathscr{A}[\ \overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}\ ]$ by $\mathscr{A}[\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.(\varphi \wedge \bar{b} = \bar{b})]$.

From the induction hypothesis, we can obtain

$$
\begin{array}{cc}\chi^{\bar{a}}_{\bar{x}},\ \forall\bar{x}.(\chi \wedge \bar{x} \prec \bar{a} \supset \mathscr{A}[\ \overline{\langle\xi\rangle\varphi}\ ]) & \\ \prod' & \sum' \\ \mathbb{TI}(\prec, \lambda\bar{x}.\chi) \quad \text{and} & \mathscr{A}[\ \overline{\langle\alpha\rangle\varphi}\ ]^{\bar{a}}_{\bar{x}}.\end{array}
$$

From $\langle\mu\rangle$ rule, the following holds:

Lemma 5.13.

$$\vdash_{\mathbf{PLi}} \mathscr{A}[\overline{\langle\alpha_\xi[\mu\xi.\alpha]\rangle\varphi}] \leftrightarrow \mathscr{A}[\overline{\langle\mu\xi.\alpha\rangle\varphi}].$$

Using this lemma, we have the following proof:

$$\dfrac{\dfrac{\prod'_{\mathbf{TI}(\prec,\lambda\bar{x}.\chi)} \chi^{\bar{a}}_{\bar{x}},\ \forall\bar{x}.(\chi \wedge \bar{x}\prec\bar{a} \supset \mathscr{A}[\overline{\langle\mu\xi.\alpha\rangle\varphi}])}{\forall\bar{x}.(\chi \supset \mathscr{A}[\overline{\langle\mu\xi.\alpha\rangle\varphi}])}\langle\bar{a}\rangle\text{IND}}{\dfrac{\sum'_{\bar{\xi}}[\overline{\mu\xi.\alpha}]\ \mathscr{A}[\overline{\langle\alpha_\xi[\mu\xi.\alpha]\rangle\varphi}]^{\bar{a}}_{\bar{x}}}{\mathscr{A}[\overline{\langle\mu\xi.\alpha\rangle\varphi}]^{\bar{a}}_{\bar{x}}}\text{Lemma 5.13}}\ \text{(IND)}\ \text{(IND)}$$

5.4. Deterministic programs

If we consider only deterministic programs, $\langle\alpha\rangle$ and $\vee$ (or $\exists\bar{x}$) becomes commutable, and Choice-I and Choice-E should be replaced by DChoice-I and DChoice-E described below, respectively. In this section, we describe a formal system for deterministic programs, $\mathbf{PLi}_D$, but only the difference from **PLi** is presented.

We need to express disjointness of boolean expressions in order to describe DChoice-I and DChoice-E rules.

$$Disj(\pi_1,\ldots,\pi_n) \triangleq \wedge_{i\neq j}\neg(\pi_i \wedge \pi_j).$$

Then new inference rules are:

Axioms and Rules 5.14.

Det-$\vee$ $\quad \dfrac{\langle\alpha\rangle \vee_i \varphi_i}{\vee_i\langle\alpha\rangle\varphi_i}$

Det-$\exists$ $\quad \dfrac{\langle\alpha\rangle\exists\bar{x}.\varphi}{\exists\bar{x}.\langle\alpha\rangle\varphi}$ $\quad$ if $\bar{x} \cap (\mathbf{LV}\alpha \cup \mathbf{RV}\alpha) = \phi$

DChoice-I $\quad \dfrac{Disj(\bar{\pi})\ \vee_i\pi_i\ \wedge_i \triangle\pi_i\ \langle\alpha_1\rangle\varphi\ \ldots\ \langle\alpha_n\rangle\varphi}{\langle\pi_1\to\alpha_1|\ldots|\pi_n\to\alpha_n\rangle\varphi}$ $\quad \begin{matrix}[\pi_1] & [\pi_n]\\ \vdots & \vdots\end{matrix}$

DChoice-E* $\quad \dfrac{Disj(\bar{\pi})\ \pi_k\ \langle\pi_1\to\alpha_1|\ldots|\pi_n\to\alpha_n\rangle\varphi}{\langle\alpha_k\rangle\varphi}$ *(for each k)*

$$\dfrac{Disj(\bar{\pi})\ \langle\pi_1\to\alpha_1|\ldots|\pi_n\to\alpha_n\rangle\varphi}{\wedge_i \triangle\pi_i \wedge \vee_i\pi_i}$$

The premise $Disj(\bar{\pi})$ in DChoice-I and DChoice-E does not impose extra constraints, because every boolean expression $\vee_i\pi_i$ can be transformed into an equivalent boolean expression for which disjointness holds:

Proposition 5.15. *Given quantifier free formulas $\pi_1,\ldots,\pi_n$, there exist quantifier free formulas $\pi'_1,\ldots,\pi'_n$ such that*

$$\vdash_{\mathrm{ULO}} Disj(\bar{\pi}'),$$

$$\vdash_{\mathrm{ULO}} \vee_i \pi_i \rightarrow \vee_i \pi'_i,$$

$$\vdash_{\mathrm{ULO}} \pi'_k \rightarrow \pi_k \quad (\textit{for each } k).$$

For $\mathbb{PLi}_{\mathrm{D}}$, Program Extraction Theorem 5.2 can be strengthened as follows:

Theorem 5.16. (Program Extraction Theorem — deterministic program version)
If we use $\mathbb{PLi}_{\mathrm{D}}$ instead of $\mathbb{PLi}$, then we can omit the condition (iii) *of Program Extraction Theorem* 5.2.

Here we make several remarks for the proof of this theorem. In the proof in the previous section, the condition (iii) is actually used only in Lemma 5.5, Lemma 5.6 and Lemma 5.11. Lemma 5.5 and Lemma 5.11 also hold for SP contexts in $\mathbb{PLi}_{\mathrm{D}}$. Moreover Lemma 5.6 holds in the following form:

Lemma 5.17. *If $\mathscr{A}[*]$ is an SP context, none of the free variables occurring in π_i's is bound in the context $\mathscr{A}[*]$ and $\mathrm{FV}\pi_i \cap \bar{y} = \phi$, the following derived rule holds in $\mathbb{PLi}_{\mathrm{D}}$:*

$$\frac{\begin{array}{c}[\pi_1]\\ \vdots\\ Disj(\bar{\pi}) \quad \vee_i \pi_i \quad \triangle\pi_1 \ldots \quad \mathscr{A}[\overline{\langle\alpha_1\rangle\varphi}]\cdots\end{array}}{\mathscr{A}[\overline{\langle\pi_1 \rightarrow \alpha_1 | \ldots | \pi_n \rightarrow \alpha_n\rangle\varphi}].}$$

Hence the proof described in Section 5.3 can similarly be applied for deterministic programs.

5.5. Underlying logic $\tilde{\exists}$-ULO

We often use ULO as underlying logic. So it is convenient to define $\tilde{\exists}$-ULO just like $\tilde{\exists}$-UL1. In $\tilde{\exists}$-ULO, $\tilde{\exists}$-I is of the following form:

Axioms and Rules 5.18.

$$\tilde{\exists}\text{-I}_0 \quad \frac{\triangle\bar{\tau} \quad \varphi_{\bar{x}}[\bar{\tau}]}{\tilde{\exists}\bar{x}.\varphi}$$

where $\bar{\tau}$ are executable and substitutible for $\bar{x}$ in φ.

Since $\tilde{\exists}$-UL1 is an extension of $\tilde{\exists}$-ULO, every proof in $\tilde{\exists}$-ULO can be transformed into a proof in $\tilde{\exists}$-UL1, and then we can apply Program Extraction Theorem to them. Hence Theorem 5.2 also holds for $\tilde{\exists}$-ULO. In this section,

we examine how proofs in $\tilde{\exists}$-**UL0** can be translated into **PLi**.

$\tilde{\exists}$-**UL0** is different from $\tilde{\exists}$-**UL1** in $\forall$-E, $\exists$-I and $\tilde{\exists}$-I. First we consider $\forall$-E_0 rule.

5.5.1. $\forall$-$\mathbf{E}_0$ Consider the following proof in $\tilde{\exists}$-**UL0**:

$$(6)\qquad \frac{\prod \quad \triangle\bar{\tau} \qquad \sum \quad \forall\bar{x}.\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]}{\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}]_{\bar{x}}[\bar{\tau}].}\ \forall\text{-}E_0$$

From the induction hypothesis, we have

$$\prod' \quad \triangle\bar{\tau} \quad \text{and} \quad \sum' \quad \forall\bar{x}.\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,].$$

We note that, instead of (6), we can construct the following proof in $\tilde{\exists}$-**UL1**:

$$\frac{\dfrac{\prod \quad \triangle\bar{\tau} \qquad \dfrac{\sum}{\forall\bar{x}.\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]}}{\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]}\ \forall\text{-E} \qquad \dfrac{\dfrac{\dfrac{}{\bar{a}=\bar{\tau}}\,(\text{let-E}) \qquad \dfrac{}{\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]^{\bar{a}}_{\bar{x}}}\,(\text{let-E})}{\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]_{\bar{x}}[\bar{\tau}]}\ \text{Substitution Lemma}}{\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]_{\bar{x}}[\bar{\tau}]}\ \langle\bar{a}\rangle\text{let-E}}$$

where $\bar{a}$ are fresh variables.

Since $\bar{x}$ do not occur in any $\tilde{\exists}$-prefix $\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m$ in view of $\tilde{\exists}$-CV,

$$\mathscr{A}[\,\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\,]_{\bar{x}}[\bar{\tau}] \equiv \mathscr{A}_{\bar{x}}[\bar{\tau}][\overline{\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi_{\bar{x}}[\bar{\tau}]}].$$

We extend Substitution Lemma for **PLi**:

Lemma 5.19. *Suppose that $\bar{x}$ are marked variables and that $\bar{a}$ are fresh variables compatible with $\bar{x}$, then the following hold:*

(i) *If $\bar{x}$ are not bound in $\mathscr{A}[*]$ and $\bar{\tau}$ are substitutible for $\bar{x}$ in $\mathscr{A}[*]$ and $\bar{\varphi}$, then*

$$\vdash_{\mathbf{PLi}} \bar{a}=\bar{\tau}\wedge\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,]^{\bar{a}}_{\bar{x}} \leftrightarrow \triangle\bar{\tau}\wedge\mathscr{A}_{\bar{x}}[\bar{\tau}][\,\overline{\langle\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]}\,].$$

(ii) *If $\bar{x}$ are bound at each $*_i$ in $\mathscr{A}[*]$ and $\bar{\tau}$ are substitutible for $\bar{x}$ in $\mathscr{A}[*]$, then*

$$\vdash_{\mathbf{PLi}} \bar{a}=\bar{\tau}\wedge\mathscr{A}[\,\overline{\langle\alpha\rangle\varphi}\,]^{\bar{a}}_{\bar{x}} \leftrightarrow \triangle\bar{\tau}\wedge\mathscr{A}_{\bar{x}}[\bar{\tau}][\,\overline{\langle\alpha\rangle\varphi}\,].$$

Hence, we have

$$\dfrac{\dfrac{\prod\limits_{\triangle\bar{\tau}}\ \forall\bar{x}.\mathscr{A}[\ \overline{\langle\alpha\rangle\varphi}\ ]}{\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\mathscr{A}[\ \overline{\langle\alpha\rangle\varphi}\ ]}\forall\text{-E} \qquad \dfrac{\overset{\text{(let-E)}}{\bar{a}=\bar{\tau}}\quad \mathscr{A}[\ \overline{\langle\alpha\rangle\varphi}\ ]^{\bar{a}}_{\bar{x}}}{\mathscr{A}_{\bar{x}}[\bar{\tau}][\ \overline{\langle\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]}\ ]}\text{Lemma 5.19 (*)}}{\mathscr{A}_{\bar{x}}[\bar{\tau}][\ \overline{\langle\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]}\ ]}\langle\bar{a}\rangle\text{let-E}$$

(*): $\bar{x}$ are not bound in $\mathscr{A}[*]$ by $\tilde{\exists}$-CV and $\bar{\tau}$ are substitutible for $\bar{x}$ in $\mathscr{A}[*]$ and φ by the condition of $\forall$-I rule.

5.5.2. $\tilde{\exists}$-I_0 For $\tilde{\exists}$-I_0 rule, consider the following proof in $\tilde{\exists}$-UL0:

$$\text{(7)}\qquad \dfrac{\prod\limits_{\triangle\bar{\tau}}\ \sum\ (\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi)_{\bar{x}}[\bar{\tau}]}{\tilde{\exists}\bar{x}\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi.}\tilde{\exists}\text{-}I_0$$

Since

$$(\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi)_{\bar{x}}[\bar{\tau}]\equiv\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.(\varphi_{\bar{x}}[\bar{\tau}]),$$

we have, from the induction hypothesis,

$$\prod_{\triangle\bar{\tau}}'\quad\text{and}\quad\sum'\langle\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}].$$

We note that (7) can be transformed into $\tilde{\exists}$-UL1 as follows:

$$\dfrac{\dfrac{\prod\limits_{\triangle\bar{\tau}}\ \dfrac{\overset{\text{(let-I)}}{\bar{a}=\bar{\tau}}\quad\sum\ (\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi)_{\bar{x}}[\bar{\tau}]}{(\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}y_m.\varphi)^{\bar{a}}_{\bar{x}}}\text{Substitution Lemma}}{\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi}\tilde{\exists}\text{-I}}{\tilde{\exists}\bar{x}\tilde{\exists}\bar{y}_1\ldots\tilde{\exists}\bar{y}_m.\varphi.}\langle\bar{a}\rangle\text{let-I}$$

where $\bar{a}$ are fresh variables. Hence we have

$$\dfrac{\dfrac{\prod\limits_{\triangle\bar{\tau}}'\ \dfrac{\overset{\text{(let-I)}}{\bar{a}=\bar{\tau}}\quad\sum'\langle\alpha\rangle\varphi_{\bar{x}}[\bar{\tau}]}{\langle\alpha\rangle\varphi^{\bar{a}}_{\bar{x}}\equiv(\langle\alpha\rangle\varphi)^{\bar{a}}_{\bar{x}}}\text{Substitution Lemma for PLi (*)}}{\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\langle\alpha\rangle\varphi}\langle\bar{a}\rangle\text{let-I}}{\langle\bar{x}:=\bar{\tau}\rangle\langle\alpha\rangle\varphi}\langle:=\rangle\text{-I}}{\langle\bar{x}:=\bar{\tau};\alpha\rangle\varphi.}\langle;\rangle$$

(*): Since $\bar{x}$ are not left variables of α from $\tilde{\exists}$-CV, Proposition 3.6(iii) can be applied.

The proof of Program Extraction Theorem for $\tilde{\exists}$-**UL0** is now completed.

5.5.3. $\forall$-**EV** We can extract a simpler program from $\forall$-E_0 if some additional condition is satisfied.

Consider the following simpler form of $\forall$-E rule which is often used:

$$\frac{\forall \bar{x}.\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}]}{\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}]^{\bar{a}}_{\bar{x}}}$$

According to the above discussion for $\forall$-E, the following is obtained:

$$\frac{\forall \bar{x}.\mathscr{A}[\overline{\langle\alpha\rangle\varphi}]}{\mathscr{A}^{\bar{a}}_{\bar{x}}[\overline{\langle \mathbf{let}\,\bar{x}\leftarrow\bar{a}\,\mathbf{in}\,\alpha\rangle\varphi^{\bar{a}}_{\bar{x}}}]}.$$

But the following lemma shows a sufficient condition for replacing **let** $\bar{x} \leftarrow \bar{a}$ **in** α simply with $\alpha^{\bar{a}}_{\bar{x}}$. However the following lemma holds:

Lemma 5.20. *If* $\bar{x}$ *are not bound in* $\mathscr{A}[\bar{*}]$, *then*

$$\vdash_{\mathrm{PLi}} \forall \bar{x}.\mathscr{A}[\overline{\langle\alpha\rangle\varphi}] \rightarrow \mathscr{A}^{\bar{a}}_{\bar{x}}[\overline{\langle\alpha^{\bar{a}}_{\bar{x}}\rangle\varphi^{\bar{a}}_{\bar{x}}}].$$

So it is convenient to separate the following special case from general $\forall$-E and to add this to $\tilde{\exists}$-**UL0** and $\tilde{\exists}$-**UL1**.

Axioms and Rules 5.21.

$$\forall\text{-EV}\quad \frac{\forall \bar{x}.\varphi}{\varphi^{\bar{a}}_{\bar{x}}}$$

For $\forall$-EV, by $\tilde{\exists}$-**CV**, $\bar{x}$ are not bound in $\mathscr{A}[\bar{*}]$, $\bar{x} \cap \bar{y} = \phi$, and

$$\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}]^{\bar{a}}_{\bar{x}} \equiv \mathscr{A}^{\bar{a}}_{\bar{x}}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}].$$

So we can use Lemma 5.20 and obtain the desired program $\alpha^{\bar{a}}_{\bar{x}}$.

Example 4. (Factorial) We consider the proof given in Example 1. In the proof we consider all the variable are marked, hence all the existential quantifiers occurring in it are marked. Thus it is a proof in $\tilde{\exists}$-**UL0**. In the sequel, we omit to present **null** programs explicitly.

First we construct subproofs corresponding to Π, Σ_1 and Σ_2:

$$\Pi' \qquad \dfrac{\dfrac{a=0 \quad \overset{FACT}{F(0,1)}}{F(a,1)}\,\text{Substitution Lemma}}{\langle z:=1\rangle F(a,z)}$$

$$\sum_1' \quad \dfrac{\dfrac{a\neq 0}{a-1<a}\,\text{PA} \qquad \dfrac{\forall x.(x<a\supset\langle\xi{:}(z;\,x)\rangle F(x,\,z))}{a-1<a\supset\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi\rangle F(a-1,\,z)}\,\forall\text{-E}}{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi\rangle F(a-1,\,z)}\,\supset\text{-E}$$

$$\sum_2' \quad \dfrac{\dfrac{\dfrac{a\neq 0}{a-1+1=a}\,\text{PA} \qquad \dfrac{F(a-1,\,b)}{F(a-1+1,(a-1+1)\cdot b)}\,\text{FACT}}{F(a,\,a\cdot b)}\,\text{Substitution Lemma}}{\langle z:=a\cdot b\rangle F(a,\,z).}$$

Then the proof Σ' corresponding to Σ becomes:

$$\dfrac{\dfrac{\dfrac{a\neq 0,\forall x.(x<a\supset\langle\xi\rangle F(x,\,z))}{\sum_1'}}{\dfrac{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi{:}(z;\,x)\rangle F(a-1,\,z)}{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi(b;\,x)\rangle F(a-1,\,b)}\,\text{Lemma 5.10}} \qquad \dfrac{\overset{(\langle\alpha\rangle\text{-rule})}{a\neq 0, F(a-1,\,b)}}{\sum_2'\ \langle z:=a\cdot b\rangle F(a,\,z)}}{\dfrac{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi{:}(b;\,x)\rangle\langle z:=a\cdot b\rangle F(a,\,z)}{\langle \mathbf{local}\, b\,\mathbf{in}((\mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi{:}(b;\,x));\ z:=a\cdot b)\rangle F(a,\,z).}\,\langle;\rangle,\text{Lemma 5.12}}\,\langle b\rangle,\langle\alpha\rangle\text{-rule}$$

Now we let:

$\alpha \triangleq z:=1$

$\beta \triangleq \mathbf{local}\ b\ \mathbf{in}\ (\mathbf{let}\ x\leftarrow x-1\ \mathbf{in}\ \xi{:}(b;\,x));\ z:=x\cdot b$

$\gamma \triangleq \mu\xi{:}(z;\,x).(x=0\rightarrow\alpha\mid x\neq 0\rightarrow\beta)$

i.e. $\mu\xi{:}\,(z;\,x).$

$(x=0\rightarrow z:=1\mid x\neq 0\rightarrow \mathbf{local}\ b\ \text{in}\ ((\mathbf{let}\ x\leftarrow x-1\ \mathbf{in}\ \xi{:}(b;\,x));\ z:=x\cdot b))$

$IH \triangleq \forall x.(x<a\supset\langle\gamma\rangle F(x,\,z)).$

Then we obtain the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a=0\vee a\neq 0}\,\text{EM} \qquad \dfrac{\overset{(\vee\text{-E})}{a=0}}{\prod'\ \langle\alpha_x^a\rangle F(a,\,z)} \qquad \dfrac{\overset{(\vee\text{-E})}{a\neq 0} \quad \overset{(\text{IND})}{IH}}{\sum'\ \langle(\beta_{x\xi}^a[\gamma]\rangle F(a,\,z)}}{\langle a=0\rightarrow\alpha_x^a\mid a\neq 0\rightarrow(\beta_x^a)_\xi[\gamma]\rangle F(a,\,z)}\,\vee\text{-E}}{\langle(a=0\rightarrow\alpha_x^a\mid a\neq 0\rightarrow\beta_x^a)_\xi[\gamma]\rangle F(a,\,z)}\,\equiv}{\langle\mu\xi_x^a.(a=0\rightarrow\alpha_x^a\mid a\neq 0\rightarrow\beta_x^a)\rangle F(a,\,z)}\,\langle\mu\rangle}{\forall x.\langle\mu\xi.(x=0\rightarrow\alpha\mid x\neq 0\rightarrow\beta)\rangle F(x,\,z).}\,\langle a\rangle,\text{IND}$$

$$\dfrac{}{\mathbb{TI}(<)}\,\text{PA}$$

5.6. Simplifying extracted programs

In the above example, undersirable renaming of variables occurs in the obtained program. In fact, it is desirable to obtain

(8) $\mu\xi{:}(z;\ x).(x = 0 \to z := 1 \mid x \neq 0 \to (\textbf{let}\ x \leftarrow x - 1\ \textbf{in}\ \xi{:}(z;\ x));\ z := x \cdot z)$

instead of

$\mu\xi{:}(z;\ x).(x = 0 \to z := 1 \mid x \neq 0 \to \textbf{local}\ u\ \textbf{in}\ (\textbf{let}\ x \leftarrow x - 1\ \textbf{in}\ \xi{:}\ (u;\ x));\ z := x \cdot u).$

This redundancy arises from our treatment of $\tilde{\exists}$-E rule.

We now consider a special case of (3) in Section 5.3.7. Suppose that $\bar{z} = \bar{x}$, $\bar{y} = \phi$ in (3) and $\mathscr{A}[\ \overline{\langle\beta\rangle\psi}\ ]$ in (4) is:

(9) $$\langle x := a + 1\rangle\psi.$$

Then we obtain the following program by the second method described in that section.

$$\textbf{local}\ u\ \textbf{in}\ (\alpha^{u}_{x};\ x := u + 1).$$

But what we really desire is:

(10) $$\alpha;\ x := x + 1.$$

To obtain this program, we first rename x to a in (9) and get

$$\langle a := a + 1\rangle\psi^{a}_{x},$$

and then we use $\langle\alpha\rangle$-rule to obtain

$$\langle\alpha;\ x := x + 1\rangle\psi.$$

This is just (10). However x is not renamable to a as in (9) in general. We must find the condition that such renaming is possible.

Definition. If $\bar{z}$ and $\bar{w}$ are compatible and $\bar{z} \subseteq \bar{x} \cup \bar{y}$, we simply write $\alpha_{\bar{z}}[\bar{w}]$ instead of $\alpha_{\bar{x};\bar{y}}[\bar{u};\ \bar{v}]$ where $\bar{u}$ and $\bar{v}$ are such that:

(i) u_i (resp. v_i) is w_i if the corresponding x_i (resp. y_i) is z_i;
(ii) u_i (resp. v_i) is x_i (resp. y_i) if x_i (resp. y_i) does not occur in $\bar{z}$.

Definition. Let x and a be compatible variables and α be a program. We say x *is identifiable to* a *in* α if $x \notin \mathbf{RV}\alpha$, $a \notin \mathbf{LV}\alpha$ and one of the following holds:

(i) x or a does not occur in α;
(ii) α is $\bar{y} := \bar{\tau}$;
(iii) α is $\textbf{null}{:}(\bar{y};\ \bar{z})$;
(iv) α is $\xi{:}(\bar{y};\ \bar{z})$ and either $x \notin \bar{y}$ or $a \notin \bar{z}$;
(v) α is $\beta;\ \beta'$, x is identifiable to a in both β and β', and either $x \notin \mathbf{LV}\beta$ or

$a \notin \mathbf{RV}\beta'$;

(vi) α is $\pi_1 \to \beta_1 | \ldots | \pi_n \to \beta_n$, x does not occur in any π_i, and x is identifiable to a in any β_i;

(vii) α is $\mu\xi{:}(\bar{y};\ \bar{z}).\beta$ and x is identifiable to a in β;

(viii) if α is **let** $\bar{y} \leftarrow \bar{\varepsilon}$ **in** β or **local** $\bar{y}$ **in** β, one of the following holds:
  (a) $x \in \bar{y}$;
  (b) x is identifiable to a in β;

(ix) if α is **var** $\bar{y}$ **in** β, the following hold:
  (a) if $x \in \bar{y}$ then $a \notin \mathbf{RV}\beta$;
  (b) x is identifiable to a in β.

Lemma 5.22. *Suppose each x_i is identifiable to a_i in α and $\bar{a}$ are substitutible to $\bar{x}$ in φ. Then the following derived rule holds:*

$$\frac{\langle\alpha\rangle\varphi}{\langle\alpha_{\bar{x}}[\bar{a}]\rangle\varphi_{\bar{x}}[\bar{a}].}$$

Now we consider the case where programs occurring in (4) in Section 5.3.7 satisfy the following conditions:

(i) $\bar{z}$ are identical to $\bar{x}$, $\bar{y}$;

(ii) for each i, x_i is identifiable to a in $\beta_\xi[\bar{y}{:}=\bar{b}]$;

(iii) for each j, y_j is identifiable to a in $\beta_\xi[\bar{y}{:}=\bar{b}]$.

Then we can extract a desired program as follows:

$$\frac{\overset{(1)}{\varphi^{\bar{a},\bar{b}}_{\bar{x},\bar{y}}}}{(\langle\bar{y}{:}=\bar{b}\rangle\varphi)^{\bar{a}}_{\bar{x}}}\ \text{Lemma 5.9}$$

$$\frac{\sum_\xi[\bar{y}{:}=\bar{b}]}{\mathscr{A}[\langle\beta_\xi[\bar{y}{:}=\bar{b}]\rangle\psi]}$$

$$\frac{\mathscr{A}[\langle\beta_\xi[\bar{y}{:}=\bar{b}]\rangle\psi]}{\mathscr{A}[\langle(\beta_\xi[\bar{y}{:}=\bar{b}])_{\bar{x},\bar{y}}[\bar{a},\bar{b}]\rangle\psi]}\ \text{Lemma 5.22}$$

$$\frac{\mathscr{A}[\langle(\beta_\xi[\bar{y}{:}=\bar{b}])_{\bar{x},\bar{y}}[\bar{a},\bar{b}]\rangle\psi]}{\mathscr{A}[\langle((\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}])^{\bar{a},\bar{b}}_{\bar{x},\bar{y}}\rangle\psi]}\ \equiv$$

$$\frac{\prod\langle\alpha\rangle\varphi \qquad \mathscr{A}[\langle((\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}])^{\bar{a},\bar{b}}_{\bar{x},\bar{y}}\rangle\psi]}{\mathscr{A}[\langle\alpha\rangle\langle(\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}]\rangle\psi]}\ \langle\bar{a},\bar{b}\rangle(1)\text{Lemma 5.11}$$

$$\frac{\mathscr{A}[\langle\alpha\rangle\langle(\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}]\rangle\psi]}{\mathscr{A}[\langle\alpha;(\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}]\rangle\psi]}\ \langle;\rangle$$

$$\frac{\mathscr{A}[\langle\alpha;(\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\bar{y}{:}=\bar{y}]\rangle\psi]}{\mathscr{A}[\langle\alpha;(\beta_{\bar{a},\bar{b}}[\bar{x},\bar{y}])_{\xi'}[\mathbf{null}{:}(\bar{y};\ \bar{y})]\rangle\psi]}\ (*)$$

where ξ' are $\xi_{\bar{b}}[\bar{y}]$.

(*): This is easily derived from $\langle$**null**$\rangle$-rule.

Example 5. (Factorial revisited) We consider the proof given in Example 1

again. Let $\prod'$, $\sum'_1$ and $\sum'_2$ be the same as Example 4. In $\sum'_2$, z is identifiable to a in $z:=a\cdot b$. So Lemma 5.22 is applicable.

Now let $\sum'$ be

$$\dfrac{\dfrac{a\neq 0,\ \forall x.(x<a\supset\langle\xi\rangle F(x,z)) \qquad \sum'_1}{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi\rangle F(a-1,z)}\ \langle\alpha\rangle\text{-rule} \qquad \dfrac{a\neq 0,\ F(a-1,b)}{\sum'_2}\ (\langle\alpha\rangle\text{-rule})}{\langle \mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi\rangle F(a-1,z)\ \langle b:=a\cdot b\rangle F(a,b)}\ \langle b\rangle,\langle\alpha\rangle\text{-rule},\langle;\rangle \qquad \dfrac{\langle z:=a\cdot b\rangle F(a,z)}{\langle b:=a\cdot b\rangle F(a,b)}\ \text{Identify } z \text{ to } b$$

$$\langle (\mathbf{let}\, x\leftarrow a-1\,\mathbf{in}\,\xi);z:=a\cdot z\rangle F(a,z)$$

Now we let:

$\alpha \triangleq z:=1$

$\beta \triangleq (\mathbf{let}\ x\leftarrow x-1\ \mathbf{in}\ \xi:(z;x));\ z:=x\cdot z$

$\gamma \triangleq \mu\xi:(z;x).(x=0\rightarrow\alpha \mid x\neq 0\rightarrow\beta)$

i.e. $\mu\xi:(z;x).(x=0\rightarrow z:=1 \mid x\neq 0\rightarrow(\mathbf{let}\ x\leftarrow x-1\ \mathbf{in}\ \xi:(z;x));\ z:=x\cdot z)$

$IH \triangleq \forall x.(x<a\supset\langle\gamma\rangle F(x,z))$.

Then we obtain the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a=0\vee a\neq 0}\ \text{PA} \qquad \dfrac{\dfrac{a=0}{\prod'}\ (\vee\text{-E}) \qquad \dfrac{a\neq 0}{\sum'}\ (\vee\text{-E}) \qquad \dfrac{}{IH}\ (\text{IND})}{\langle\alpha^a_x\rangle F(a,z)\ \langle(\beta^a_x)_\xi[\gamma]\rangle F(a,z)}\ \vee\text{-E}}{\langle a=0\rightarrow\alpha^a_x \mid a\neq 0\rightarrow(\beta^a_x)_\xi[\gamma]\rangle F(a,z)}\equiv}{\langle(a=0\rightarrow\alpha^a_x \mid a\neq 0\rightarrow\beta^a_x)_\xi[\gamma]\rangle F(a,z)}\ \langle\mu\rangle}{\langle\mu\xi^a_x.(a=0\rightarrow\alpha^a_x \mid a\neq 0\rightarrow\beta^a_x)\rangle F(a,z)}\ \langle\alpha\rangle,\text{IND}}{\forall x.\langle\mu\xi.(x=0\rightarrow\alpha \mid x\neq 0\rightarrow\beta)\rangle F(x,z).}\ \dfrac{}{\mathbf{TI}(<)}\ \text{PA}$$

Thus we obtain the program (8).

§6. Extension of Program Extraction Theorem

6.1. Extending the Program Extraction Theorem

We extracted two factorial programs from the same proof: programs in Examples 4 and 5. But the program obtained in Section 5.6 does not correspond to the original proof straightforwardly. So we consider what kind of proof directly corresponds to the program (8).

In (8), z occurs as both a left variable and a right variable in the portion of $z:=x\cdot z$. If we keep the requirement of $\tilde{\exists}$-**CV**, this cannot happen. Hence it is necessary to extend Program Extraction Theorem by allowing formulas and sequents such that

(11) $\forall \bar{x}.(\varphi \supset \tilde{\exists} x.\psi)$ and $\Gamma[\bar{x}] \to \tilde{\exists} x.\psi$.

However, this causes another problem. As a result of this modification, left variables occurring in the extracted programs may occur as right variables. In (11), the p-types corresponding to $\tilde{\exists}\bar{x}.\psi$ have $\bar{x}$ as left and right variables; hence $\bar{x}$ are not pure left variables any longer.

In the proof described in Section 5, we use the fact $\bar{x}$ corresponding $\tilde{\exists}\bar{x}.\psi$ are pure left variables in the following points:

- Since pure left variables can be considered as bound variables by) Lemma 3.5, the eigen variable conditions for the original proof inherits to the transformed one.
- If renaming or substitution is used to describe the original proof, the same renaming or substitution can be applied to the transformed one. For example, if $\mathscr{A}^{\bar{a}}_{\bar{x}}[\langle\alpha\rangle\varphi]$ corresponds to $\mathscr{A}[\tilde{\exists}\bar{x}.\varphi]^{\bar{a}}_{\bar{x}}$ ($\equiv \mathscr{A}^{\bar{a}}_{\bar{x}}[\tilde{\exists}\bar{x}.\varphi]$), $\mathscr{A}[\langle\alpha\rangle\varphi]$ can be considered to correspond to $\mathscr{A}[\tilde{\exists}\bar{x}.\varphi]$.
- For the rules in which $\tilde{\exists}$-formulas may be discharged (let-E, $\tilde{\exists}$-E and Ind($\prec$, $\lambda\bar{x}.\chi$) rules), we adjust the p-types of the programs in the discharged formulas by Lemma 5.4.

Therefore special conditions are required for the rules to which the eigen variable conditions for marked variables are associated, for the rules in which renaming of marked variables occurs and for the rules in which some $\tilde{\exists}$-formulas may be discharged. The extended version of Program Extraction Theorem is as follows:

Theorem 6.1. (Program Extraction Theorem—Revised) *Program Extraction Theorem* 5.2 *also holds for the following changes:*

Delete the condition (i).

Add the following conditions for marked variables: (*in the following, variables* x_i *and* a_i *range over only marked variables*)

(iv) *For each rule with which the eigen variable condition is associated: If* $\bar{a}$ *are the eigen variables corresponding to* $\bar{x}$, *and* a_i *is in* $\bar{x}$, *then* a_i *must be just the corresponding* x_i.

(v) *For* $\vee$-E: *If* $\tilde{\exists}$-*formulas occur in the minor premise,* $\vee_i\pi_i$ *is the major premise and* $\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}]$ *is the conclusion, then*

(a) *all the variables occurring free in* $\vee_i\pi_i$ *must not be bound in* $\mathscr{A}[\bar{*}]$.

(vi) *For* **let**-I: *If* **let** $\bar{x} \leftarrow \bar{\tau}$ **in** $\mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}]$ *is the conclusion and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *if there is* $*_j$ *such that* x_i *is not bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then* x_i *does not occur free in any assumption* (*including the discharged one*);

(b) *if there is* $*_j$ *such that* x_i *is bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then the eigen variable* a_i *corresponding to* x_i *must be just* x_i;

(c) *if the eigen variable* a_i *corresponding to* x_i *is bound at some* $*_j$ *in* $\mathscr{A}[\bar{*}]$ *and* $a_i \not\equiv x_i$, *then* $x_i \notin \mathbf{FV}(\mathscr{A}[\bar{*}])$.

(vii) *For* **let**-E: *If* **let** $\bar{x} \leftarrow \bar{\tau}$ **in** $\mathscr{A}[\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m . \varphi}]$ *is the major premise,* $\mathscr{B}[\overline{\exists \bar{z}_1 \ldots \exists \bar{z}_n . \psi}]$ *is the minor premise, and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *for each variable* u *occurring free in* $\bar{\tau}$ *or in the assumptions above the major premise, if* u *is not bound at* $*_j$ *in* **let** $\bar{x} \leftarrow \bar{\tau}$ **in** $\mathscr{A}[\bar{*}]$, *then* $u \in \mathbf{FV}(\mathscr{A}[\bar{*}])$ *or for* $\bar{\exists}$*-prefix formula* $\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi$ *corresponding* $*_j$, $u \in \mathbf{FV}(\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi)$;

(b) *if* x_i *is not bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then* $x_i \in \mathbf{FV}(\mathscr{A}[\bar{*}])$ *or for* $\bar{\exists}$*-prefix formula* $\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi$ *corresponding* $*_j$, $u \in \mathbf{FV}(\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi)$;

(c) *if an eigen variable* a_i *is not bound at* $*_j$ *in* $\mathscr{B}[\bar{*}]$, *then the corresponding* τ_i *must be substitutible at* $*_j$ *in* $\mathscr{B}[\bar{*}]$.

(viii) *For* $\forall$-I: *If* $\forall \bar{x} . \mathscr{A}[\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m . \varphi}]$ *is the conclusion and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *if there is* $*_j$ *such that* x_i *is not bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then* x_i *does not occur free in any assumption*;

(b) *if there is* $*_j$ *such that* x_i *is bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then the eigen variable* a_i *corresponding to* x_i *must be just* x_i;

(c) *if the eigen variable* a_i *corresponding to* x_i *is bound at some* $*_j$ *in* $\mathscr{A}[\bar{*}]$ *and* $a_i \not\equiv x_i$, *then* $x_i \notin \mathbf{FV}(\mathscr{A}[\bar{*}])$.

(ix) *For* $\forall$-EV: *If* $\forall \bar{x} . \mathscr{A}[\overline{\exists \bar{y}_1 \ldots \exists \bar{y}_m . \varphi}]$ *is the premise,* $\bar{a}$ *are the variables substituted for* $\bar{x}$ *and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *if* $x_i \in \bar{y}_1 \cup \ldots \cup \bar{y}_n$ *or* x_i *occurs free in the assumptions, then* $a_i \equiv x_i$.

(x) *For* $\exists$-E and $\bar{\exists}$-E: *If* $\exists \bar{x} . \bar{\exists} \bar{y}_1 \cdots \bar{\exists} \bar{y}_m . \varphi$ (*or* $\bar{\exists} \bar{x} . \bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi$) *is the major premise,* $\mathscr{A}[\overline{\exists \bar{z}_1 \ldots \exists \bar{z}_n . \psi}]$ *is the minor premise and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *all the free variables occurring in the assumptions above the major premise must occur free in* $\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi$.

(xi) *For* $\mathrm{Ind}(\prec, \lambda \bar{x} . \chi)$: *If* $\mathscr{A}[\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi]$ *is the premise other than* $\mathrm{TI}(\prec, \lambda \bar{x} . \chi)$, Γ *is the set of assumptions above the* $\mathscr{A}[\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi]$, *and* $\mathscr{A}[\bar{*}]$ *has at least one* $*$, *then*

(a) *for each variable* u *occurring free in* Γ, *if* u *is not bound at* $*_j$ *in* $\mathscr{A}[\bar{*}]$, *then* $u \in \mathbf{FV}(\mathscr{A}[\bar{*}])$ *or for* $\bar{\exists}$*-prefix formula* $\bar{\exists} \bar{y}_1 \ldots \bar{\exists} \bar{y}_m . \varphi$ *corresponding* $*_j$,

$u \in \mathbb{FV}(\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\, \varphi)$;

(b) *for each* x_i, x_i *does not occur free in* Γ;

(c) *for each* x_i, x_i *and the corresponding eigen variable* a_i *are not bound in* $\mathscr{A}[*]$.

We note that this theorem includes Theorem 5.2, because the conditions (iv), (vi), (vii), (viii), (x) and (xi) are derived from the condition $\tilde{\exists}$-$\mathbb{C}\mathbb{V}$ and we can obtain the equivalent formula of φ with additional free variables $\bar{u}$ as $\bar{u} = \bar{u} \wedge \varphi$.

The conditions for **let**-I and **let**-E are slightly strong. For example, the proof in $\tilde{\exists}$-$\mathbb{UL}\mathbb{1}$ corresponding to

$$\dfrac{\varphi[x+z+1,\ y]}{\tilde{\exists}x.\varphi[x,\ y]}\,\tilde{\exists}\text{-I}_0$$

(which uses **let**-I) may satisfy the condition (vi), but the proof in $\tilde{\exists}$-$\mathbb{UL}\mathbb{1}$ corresponding to

$$\dfrac{\tilde{\exists}y.\varphi[x+z+1,\ y]}{\tilde{\exists}x.\tilde{\exists}y.\varphi[x,\ y]}\,\tilde{\exists}\text{-I}_0 \tag{12}$$

need not satisfy the condition, because it uses the following instance of **let**-I:

$$\dfrac{\begin{array}{c}[a = x+z+1]\\ \vdots\\ \tilde{\exists}y.\varphi[a,\ y]\end{array}}{\textbf{let}\, x \leftarrow x+z+1\, \textbf{in}\, \tilde{\exists}y.\varphi[x,\ y]}\,\langle a\rangle\,\textbf{let-I} \tag{13}$$

which violates the condition (vi-a). Hence if we need the derivation corresponding to (12), we must include $\tilde{\exists}$-I_0 in the primitive rules of $\tilde{\exists}$-$\mathbb{UL}\mathbb{1}$. By the same reason, we include $\forall$-E_0 in our consideration.

Now we consider the following condition $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$:

Definition. A $\tilde{\exists}$-SNP formula satisfies *weak* $\tilde{\exists}$-$\mathbb{C}\mathbb{V}$ (denoted by $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$) if

- ⊙ it is obtained from an SNP-context $\mathscr{A}[*]$ which satisfies (strong) $\mathbb{C}\mathbb{V}$ for marked variables, and
- ⊗ for every $\tilde{\exists}$-prefix $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n$ occurring in it, $\bar{x}_i$'s are mutually disjoint.

A $\tilde{\exists}$-SNP sequent satisfies $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$ if the universal closure of $\wedge\Gamma \supset \varphi$ satisfies $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$. A $\tilde{\exists}$-SNP proof satisfies $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$ if all the constituent sequents of it satisfy $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$.

For example, $\forall x,\ y.(F(x,\ y) \supset (\tilde{\exists}x.\, G(x,\ y)))$ satisfies $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$.

It is obvious that $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$ admits formulas and sequents of the form (11). On the other hand, the proof of the form (13) does not satisfy $\tilde{\exists}$-w$\mathbb{C}\mathbb{V}$, but

we already know that it is excluded from our consideration. So the class of $\tilde{\exists}$-wCV proofs is not so restrictive in our purpose. If we restrict proofs to $\tilde{\exists}$-wCV, then the conditions (vi-c) and (viii-c) are automatically satisfied. The condition (v-a) is also satisfied if the discharged assumptions π_i's are actually used.

6.2. Proof of Extended Theorem

Suppose

$$\sum \vdash_{\tilde{\exists}\text{-UL1}} \Gamma[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\, \varphi}] \to \mathscr{A}[\overline{\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\, \psi}]. \tag{14}$$

In what follows, we shall construct a proof Σ' of **PLi** from (14) such that it satisfies the following additional property:

(B) if $\mathscr{A}[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\, \varphi}]$ occurs as an assumption in Σ and is transformed into $\mathscr{A}[\overline{\langle\xi\rangle\varphi}]$ in Σ', and x in $\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n$ occurs as a right variable of some ξ_i's, then x occurs free in $\mathscr{A}[*]$.

So if x is an eigen variable of some inference which has $\mathscr{A}[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\, \varphi}]$ as an assumption, then x is not free in $\mathscr{A}[\overline{\tilde{\exists}\bar{x}_1 \ldots \tilde{\exists}\bar{x}_n.\, \varphi}]$; hence x is a pure left variable in $\mathscr{A}[\overline{\langle\xi\rangle\varphi}]$. Thus x can be regarded as a bound variable. So our transformation does not violate eigen variable conditions as far as assumptions are concerned.

We prove the theorem by induction on the length of proofs in the same way as in Section 5.3. Base case is proved in the same way as 5.3.1.

We here consider only the typical rules for which an eigen variable condition is associated, substitution occurs (for **UL0** specific rules), or renaming occurs. Moreover we only consider the case where marked variables are essential. The other case are easily established. For the simplicity, we shall omit to mention that we use Lemma 5.5 to reduce right variables of the obtained program when it has extra right variables. We shall also omit to mention that we use Lemma 5.4 in order to adjust the p-type of the obtained program.

6.2.1. let-I Assume that the proof is of the following form:

$$\frac{\prod_{\triangle\tau} \quad \begin{matrix}[a=\tau] \\ \sum \\ \mathscr{A}[\tilde{\exists}\bar{y}.\, \varphi]^a_x\end{matrix}}{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\tilde{\exists}\bar{y}.\, \varphi].}\ \langle a\rangle$$

From Π part, using the induction hypothesis, we have

$$\prod'_{\triangle\tau.}$$

First we consider the simple case where $a \equiv x$: Since $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]$, we have, from the induction hypothesis,

$$\overset{x \equiv \tau}{\sum'} \mathscr{A}[\langle\alpha\rangle\varphi].$$

Hence we have the following proof:

$$\dfrac{\prod\limits_{\tau}' \dfrac{\overset{[x=\tau]}{\sum'} \mathscr{A}[\langle\alpha\rangle\varphi]}{\mathscr{A}[\langle\alpha\rangle\varphi]^x_x} \equiv}{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\langle\alpha\rangle\varphi].}\ \langle x\rangle\ \text{let-I}$$

If $x \in \mathbb{RV}\alpha$, then α has the desired p-type. If $x \notin \mathbb{RV}\alpha$, then x is added as a right variable at follows:

$$\dfrac{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\langle\alpha\rangle\varphi]}{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\langle \mathbf{null}\!:\!(;x);\alpha\rangle\varphi],}\ \langle \mathbf{null}\rangle$$

where $\mathbf{null}\!:\!(;x);\alpha$ has the desired p-type.

Next we assume $a \not\equiv x$. We consider the following five cases:

(I) x is bound in $\mathscr{A}[*]$;

(II) x is not bound in $\mathscr{A}[*]$, and a is bound in $\mathscr{A}[*]$;

(III) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, $x \notin \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$, and the discharged assumption $a = \tau$ is not used;

(IV) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, either $x \in \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$ or the discharged assumption $a = \tau$ is actually used, and $x \notin \bar{y}$;

(V) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, either $x \in \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$ or the discharged assumption $a = \tau$ is actually used, and $x \in \bar{y}$.

Ad. (I): From the condition (vi-b), the eigen variable a must be just x; hence this case does not occur.

Ad. (II): By the condition (vi-c), $x \notin \mathbb{FV}(\mathscr{A}[\bar{*}])$. Moreover, from the condition for renaming, $x \notin \mathbb{FV}(\tilde{\exists}\bar{y}.\varphi)$; so $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]$. Hence from the induction hypothesis, we have

$$\overset{a = \tau}{\sum'} \mathscr{A}[\langle\alpha\!:\!(\bar{y};\ \bar{u})\rangle\varphi].$$

By (vi-a), $x \notin \bar{u}$; so if $x \in \bar{y}$, x is a pure left variables, hence x is regarded as a bound variable. Hence

$$\mathscr{A}[\langle\alpha{:}(\bar{y};\ \bar{u})\rangle\varphi] \leftrightarrow \mathscr{A}[\langle\alpha{:}\ (\bar{y};\bar{u})\rangle\varphi]^a_x$$

and

$$\mathscr{A}[\langle\alpha\rangle\varphi]^a_x \leftrightarrow \mathscr{A}[\langle\mathbf{null}{:}(;\ x);\ \alpha\rangle\varphi]^a_x.$$

So we have,

$$\dfrac{\prod\limits_{\tau}' \dfrac{\begin{array}{c}[a=\tau]\\ \sum'\\ \mathscr{A}[\langle\alpha\rangle\varphi]\end{array}}{\mathscr{A}[\langle\mathbf{null}{:}(;x);\alpha\rangle\varphi]^a_x}}{\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,\mathscr{A}[\langle\mathbf{null}{:}(;x);\alpha\rangle\varphi].}\langle a\rangle\text{let-I}$$

Ad. (III): Clearly $\mathscr{A}[\tilde{\exists}\bar{y}.\,\varphi]^a_x \equiv \mathscr{A}[\tilde{\exists}\bar{y}.\,\varphi]$; so from the induction hypothesis we have

$$\sum'\mathscr{A}^a_x[\langle\alpha\rangle\varphi]$$

We also have $a\notin\mathbb{FV}(\mathscr{A}[\langle\alpha\rangle\varphi])$, especially $a\notin\mathbf{RV}\alpha$. By (vi-a), $x\notin\mathbf{RV}\alpha$, hence

$$\mathscr{A}[\langle\alpha\rangle\varphi]^a_x \leftrightarrow \mathscr{A}[\langle\alpha\rangle\varphi]$$

and

$$\mathscr{A}[\langle\mathbf{null}{:}(;\ x);\ \alpha\rangle\varphi]^a_x \leftrightarrow \mathscr{A}[\langle\alpha\rangle\varphi]^a_x.$$

So we have,

$$\dfrac{\prod\limits_{\tau}' \dfrac{\begin{array}{c}\sum'\\ \mathscr{A}[\langle\alpha\rangle\varphi]\end{array}}{\mathscr{A}[\langle\mathbf{null}{:}(;x);\alpha\rangle\varphi]^a_x}}{\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,\mathscr{A}[\langle\mathbf{null}{:}(;x);\alpha\rangle\varphi].}\langle a\rangle\text{let-I}$$

Ad. (IV): In this case, $\mathscr{A}[\tilde{\exists}\bar{y},\varphi]^a_x \equiv \mathscr{A}^a_x[\tilde{\exists}\bar{y}.\,\varphi^a_x]$; so from the induction hypothesis, we have

$$\sum_{a\equiv\tau}'\mathscr{A}^a_x[\langle\alpha{:}\ (\bar{y};\ a,\ \bar{u})\rangle\varphi^a_x].$$

By (vi-a), $x\notin\bar{u}$; so

$$\mathscr{A}^a_x[\langle\alpha{:}\ (\bar{y};\ a,\ \bar{u})\rangle\varphi^a_x] \equiv \mathscr{A}[\langle\alpha^x_a{:}\ (\bar{y};\ x,\ \bar{u})\rangle\varphi]^a_x.$$

Thus we have

$$\dfrac{\prod\limits_{\tau}{}' \dfrac{\overset{[a=\tau]}{\sum'}\mathscr{A}^a_x[\langle\alpha\rangle\varphi^a_x]}{\mathscr{A}[\langle\alpha^x_a\rangle\varphi]^a_x} \equiv}{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\langle\alpha^x_a\rangle\varphi].}\ \langle a\rangle\text{let-I}$$

Ad. (V): In this case, $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}^a_x[\tilde{\exists}\bar{y}.\varphi]$. Let $\bar{y} = x, \bar{y}'$, then from the induction hypothesis, we have

$$\overset{a \equiv \tau}{\sum'} \mathscr{A}^a_x[\langle\alpha\colon (x, \bar{y}'; a, \bar{u})\rangle\varphi]$$

where $x \notin \bar{u}$ by the condition (vi-a). Since both x and a occur in α, we cannot rewrite α as $(\alpha^x_a)^a_x$; so we cannot apply **let**-I straightforwardly. Here it suffices to consider the case where $a \not\equiv x$.

In this case, x is a pure left variable in the program $\alpha\colon (x, \bar{y}'; a, \bar{u})$; so we can rename x in α. That is, let w be a variable not occurring in $\langle\alpha\rangle\varphi$, then

$$\mathscr{A}^a_x[\langle\alpha\colon (x, \bar{y}'; a, \bar{u})\rangle\varphi] \leftrightarrow \mathscr{A}^a_x[\langle\alpha^w_x\colon (w, \bar{y}'; a, \bar{u})\rangle\varphi^w_x]. \tag{15}$$

Here we remark the following lemma:

Lemma 6.2. *If* $w \notin \mathbb{FV}(\langle\alpha\rangle\varphi)$, *then*

$$\langle\alpha\rangle\varphi \leftrightarrow \langle\alpha; w := x\rangle\varphi^w_x.$$

By this lemma,

$$\mathscr{A}^a_x[\langle\alpha^w_x\colon (w, \bar{y}'; a, \bar{u})\rangle\varphi^w_x] \leftrightarrow \mathscr{A}^a_x[\langle\alpha^w_x\colon (w, \bar{y}'; a, \bar{u}); a := w\rangle(\varphi^w_x)^a_w]. \tag{16}$$

Now let β be $\alpha^w_x; a := w$, then $w \in \mathfrak{pLV}\,\beta$, and $w \notin \mathbb{FV}((\varphi^w_x)^a_w) = \mathbb{FV}(\varphi^a_x)$. So by Lemma 5.12,

$$\mathscr{A}^a_x[\langle\beta\rangle\varphi^a_x] \leftrightarrow \mathscr{A}^a_x[\langle\mathbf{local}\ w\ \mathbf{in}\ \beta\rangle\varphi^a_x]. \tag{17}$$

Thus we have

$$\dfrac{\prod\limits_{\tau}{}' \dfrac{\dfrac{\overset{[a=\tau]}{\sum'}\mathscr{A}^a_x[\langle\alpha\rangle\varphi]}{\mathscr{A}^a_x[\langle\mathbf{local}\, w\, \mathbf{in}\, \beta\rangle\varphi^a_x]}\ \text{using(15),(16)and(17)} \quad \equiv}{\mathscr{A}[\langle\mathbf{local}\, w\, \mathbf{in}\, \beta^x_a\rangle\varphi]^a_x}}{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, A[\langle\mathbf{local}\, w\, \mathbf{in}\, \beta^x_a\rangle\varphi].}\ \langle a\rangle\text{let-I}$$

We note that the p-types of **local** w in β^x_a is $(x, \bar{y}; x, \bar{u})$ as desired.

6.2.2. let-E Assume that the proof is of the following form:

(18)
$$\dfrac{\prod\limits_{\mathbf{let}\, x \leftarrow \tau\, \mathbf{in}\, \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]} \quad \dfrac{\overset{\text{(let-E)}}{a=\tau} \quad \overset{\text{(let-E)}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x}}{\sum\limits_{\mathscr{B}[\tilde{\exists}\bar{z}.\psi]}}}{\mathscr{B}[\tilde{\exists}\bar{z}.\psi].}\ \langle a\rangle\text{let-E}$$

Let Γ be the assumptions of Π and $(\bar{y};\ x,\ \bar{u})$ be the p-type of $\tilde{\exists}\bar{y}.\,\varphi$ in the conclusion $\mathbf{let}\ x \leftarrow \tau\ \mathbf{in}\ \mathscr{A}[\tilde{\exists}\bar{y}.\,\varphi]$ of Π.

For each $u_i \in \bar{u}$,

- if $u_i \equiv a$, then $a \not\equiv x$ and, from the condition (vii-a), a is bound in $\mathscr{A}[*]$; hence a is bound in $\mathscr{A}[*]^a_x$;
- if $u_i \not\equiv a$ and u_i is bound in $\mathscr{A}[*]$, the u_i is bound in $\mathscr{A}[*]^a_x$;
- if $u_i \not\equiv a$ and u_i is not bound in $\mathscr{A}[*]$, then $u_i \in \mathbf{FV}\Gamma \cup \mathbf{FV}\tau$; hence from the condition (vii-a), $u_i \in \mathbf{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\,\varphi])$; so $u_i \in \mathbf{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x)$;
- if $u_i \in \mathbf{FV}\varphi$ and $u_i \notin \bar{y}$, then $u_i \in \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x$.

From this consideration, we know that, for the assumption $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x$ of $\sum$, we can choose a program variable ξ corresponding to $\tilde{\exists}\bar{y}.\varphi$ such that $\bar{u} \subseteq \mathbf{RV}\,\xi$.

Now from the induction hypothesis, we have

$$\prod\nolimits'_{\mathbf{let}\ x \leftarrow \tau\ \mathbf{in}\ \mathscr{A}[\langle\alpha:\,(\bar{y};x,\bar{u})\rangle\varphi]}.$$

First we assume that we could obtain the following proof:

(19)
$$\dfrac{a=\tau \quad \mathscr{A}[\langle\xi:(\bar{y};x,\bar{u})\rangle\varphi]^a_x \quad \sum\nolimits'_{\mathscr{B}[\langle\beta:(\bar{z};\bar{v})\rangle\psi]}}{\mathscr{B}[\langle\beta:(\bar{z};\bar{v})\rangle\psi].}$$

Let b be a fresh variable, and w be follows:

$$w \equiv \begin{cases} a & \text{if } x \in \bar{z} \cup \bar{v} \\ x & \text{otherwise,} \end{cases}$$

then $\beta_\xi[\alpha] \equiv ((\beta^w_a)_\xi[\alpha])^a_w$. Hence if a is not bound in $\mathscr{B}[*]$, we have

$$\dfrac{\prod\nolimits'_{\mathbf{let}\ x \leftarrow \tau\ \mathbf{in}\ \mathscr{A}[\langle\alpha\rangle\varphi]} \quad \dfrac{\overset{\text{(let-E)}}{b=\tau} \quad \dfrac{\overset{(1)}{a=\tau} \quad \dfrac{\dfrac{\overset{(1)}{a=\tau} \quad \overset{\text{(let-E)}}{b=\tau}}{a=b} \quad \overset{\text{(let-E)}}{\mathscr{A}[\langle\alpha\rangle\varphi]^b_x}}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}\ \text{Rename}}{\sum\nolimits'_\xi[\alpha]_{\mathscr{B}[\langle\beta_\xi[\alpha]\rangle\psi]}}}{\triangle\tau \quad \mathscr{B}[\langle(\beta^w_a)_\xi[\alpha]\rangle^a_w\psi]}\ \equiv}{\mathscr{B}[\langle\mathbf{let}\, w \leftarrow \tau\ \mathbf{in}(\beta^w_a)_\xi[\alpha]\rangle\psi]}\ \langle a\rangle\ (1)\ \text{Lemma}\,5.7(*)$$
$$\dfrac{}{\mathscr{B}[\langle\mathbf{let}\, w \leftarrow \tau\ \mathbf{in}(\beta^w_a)_\xi[\alpha]\psi].}\ \langle b\rangle\text{let-E}$$

(*): since a is the eigen variable in the original proof, $a \notin \mathbb{FV}\psi \cup \mathbb{FV}((\beta_a^w)_\xi[\alpha])$, and τ is substitutible by the condition (vii-c); hence Lemma 5.7 can be applied.

When a is bound in $\mathscr{B}[*]$, a appears as a right variable of the extracted program. So the following simple proof is sufficient:

$$\dfrac{\begin{array}{cc} & \overset{\text{(let-E)}}{a=\tau\ \ \mathscr{A}[\langle\alpha\rangle\varphi]_x^a} \\ \prod' & \sum'_\xi[\alpha] \\ \textbf{let}\, x \leftarrow \tau\, \textbf{in}\, \mathscr{A}[\langle\alpha\rangle\varphi] & \mathscr{B}[\langle\beta_\xi[\alpha]\rangle\psi] \end{array}}{\mathscr{B}[\langle\beta_\xi[\alpha]\rangle\psi].}\ \langle b\rangle\text{let-E}$$

Thus we obtain the proof corresponding to (18).

To complete the proof for the **let**-E case, it still remains to construct a proof Σ' and a program β satisfying (19). We consider the following five cases:

(I) x is bound in $\mathscr{A}[*]$;
(II) x is not bound in $\mathscr{A}[*]$, and a is bound in $\mathscr{A}[*]$;
(III) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, and $x \notin \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$;
(IV) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, $x \in \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$, and $x \notin \bar{y}$;
(V) x is not bound in $\mathscr{A}[*]$, a is not bound in $\mathscr{A}[*]$, $x \in \mathbb{FV}(\mathscr{A}[\tilde{\exists}\bar{y}.\varphi])$, and $x \in \bar{y}$.

Ad. (I): Since $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x^a \equiv \mathscr{A}_x^a[\tilde{\exists}\bar{y}.\varphi]$, from the induction hypothesis, we have

$$a = \tau\ \ \mathscr{A}_x^a[\langle\eta\colon(\bar{y}; x, \bar{u})\rangle\varphi] \quad \sum'' \quad \mathscr{B}[\langle\beta'\colon(\bar{z}; \bar{v})\rangle\psi].$$

We also have

$$\mathscr{A}_x^a[\langle\eta\colon(\bar{y};\, x,\, \bar{u})\rangle\varphi] \equiv \mathscr{A}[\langle\eta\colon(\bar{y};\,(x,\,\bar{u})\rangle\varphi]_x^a.$$

Hence we let $\beta \equiv \beta'_\eta[\xi]$ and Σ' be

$$a=\tau\ \ \dfrac{\mathscr{A}[\langle\xi\rangle\varphi]_x^a}{\mathscr{A}_x^a[\langle\xi\rangle\varphi]} \equiv \quad \sum''_\eta[\xi] \quad \mathscr{B}[\langle\beta'\rangle\psi].$$

Ad. (II): By the condition for renaming, $x \notin \mathbb{FV}(\exists\bar{y}.\varphi)$. So $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x^a \equiv \mathscr{A}_x^a[\tilde{\exists}\bar{y}.\varphi]$. The rest can be handled in the same way as the case (I).

Ad. (III): By the condition (vii-b), this case cannot happen.

Ad. (IV): In this case, $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x^a \equiv \mathscr{A}_x^a[\tilde{\exists}\bar{y}.\varphi_x^a]$; so, from the induction hypothesis, we have

$$a = \tau \quad \mathscr{A}^a_x[\langle \eta\colon (\bar{y}; a, \bar{u})\rangle \varphi^a_x] \quad \sum'' \quad \mathscr{B}[\langle \beta'\colon (\bar{z}; \bar{v})\rangle \psi].$$

We also have

$$\mathscr{A}^a_x[\langle \eta\colon (\bar{y};\, a,\, \bar{u})\rangle \varphi^a_x] \equiv \mathscr{A}[\langle \eta^x_a\colon (\bar{y};\, x,\, \bar{u})\rangle \varphi]^a_x.$$

Hence we let $\beta' \equiv \beta_\eta[\xi^a_x]$ and Σ' be

$$a=\tau \quad \frac{\mathscr{A}[\langle \xi\rangle \varphi]^a_x}{\mathscr{A}^a_x[\langle \xi^a_x\rangle \varphi^a_x]} \equiv \quad \sum''_\eta[\xi^a_x] \quad \mathscr{B}[\langle \beta'\rangle \psi].$$

Ad. (V): In this case, $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}^a_x[\tilde{\exists}\bar{y}.\varphi]$. Let $\bar{y} = x, \bar{y}'$, then from the induction hypothesis, we have

$$a = \tau \quad \mathscr{A}^a_x[\langle \eta\colon (x,\, \bar{y}'; a, \bar{u})\rangle \varphi] \quad \sum'' \quad \mathscr{B}[\langle \beta'\colon (\bar{z}; \bar{v})\rangle \psi].$$

We proceed by separating cases depending on whether $a \equiv x$ or not.

When $a \equiv x$,

$$\mathscr{A}[\langle \eta\colon (\bar{y}; x, \bar{u})\rangle \varphi]^a_x \equiv \mathscr{A}[\langle \eta\colon (\bar{y}; x, \bar{u})\rangle \varphi]^x_x \equiv \mathscr{A}^x_x[\langle \eta\colon (\bar{y}; x, \bar{u})\rangle \varphi] \equiv \mathscr{A}^a_x[\langle \eta\colon (\bar{y}; a, \bar{u})\rangle \varphi].$$

Hence we can simply let $\beta \equiv \beta'_\eta[\xi]$ and Σ' be $\sum''_\eta[\xi]$.

When $a \not\equiv x$,

$$\mathscr{A}[\langle \xi\colon (x,\, \bar{y}';\, x,\, \bar{u})\rangle \varphi]^a_x \leftrightarrow \mathscr{A}^a_x[\langle \xi^a_x\colon (a,\, \bar{y}';\, a,\, \bar{u})\rangle \varphi^a_x]. \tag{20}$$

Here we need the following lemma.

Lemma 6.3. *If* $u \notin \mathbf{FV}(\langle \alpha\rangle \varphi)$, *then*

$$\langle \alpha\rangle \varphi \leftrightarrow \langle u := x; \alpha^u_x\rangle \varphi^u_x.$$

From this lemma,

$$\mathscr{A}^a_x[\langle \xi^a_x\colon (a,\, \bar{y}';\, a,\, \bar{u})\rangle \varphi^a_x] \leftrightarrow \mathscr{A}^a_x[\langle x := a;\, \xi\colon (x,\, \bar{y}';\, x,\, \bar{u})\rangle \varphi].$$

Using (20),

$$\mathscr{A}[\langle \xi\colon (x,\, \bar{y}';\, x,\, \bar{u})\rangle \varphi]^a_x \leftrightarrow \mathscr{A}^a_x[\langle x := a;\, \xi\colon (x,\, \bar{y}';\, x,\, \bar{u})\rangle \varphi].$$

Since the p-type of $x := a;\, \xi$ is $(x,\, \bar{y}';\, a,\, \bar{u})$ that is the p-type of η, we can let β be $\beta'_\eta[x := a;\, \xi]$ and Σ' be

$$a=\tau \quad \frac{\mathscr{A}[\langle \xi\rangle \varphi]^a_x}{\mathscr{A}^a_x[\langle x := a; \xi\rangle \varphi]} \quad \sum''_\eta[x := a; \xi] \quad \mathscr{B}[\langle \beta'\rangle \psi].$$

6.2.3. $\forall$-I This case is quite similar to **let**-I, so we omit the detailed proof.

6.2.4. $\forall$-EV Consider the following proof:

$$\dfrac{\sum \quad \forall x..\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x.}$$

From the induction hypothesis, we have

$$\sum' \quad \forall x..\mathscr{A}[\langle\alpha\rangle\varphi].$$

Now we consider the following three cases:

(I) x is bound in $\mathscr{A}[*]$;
(II) x is not bound in $\mathscr{A}[*]$, and $x \notin \bar{y}$;
(III) x is not bound in $\mathscr{A}[*]$, and $x \in \bar{y}$.

Ad. (I): Since

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}^a_x[\tilde{\exists}\bar{y}.\varphi] \quad \text{and} \quad \mathscr{A}[\langle\alpha\rangle\varphi]^a_x \equiv \mathscr{A}^a_x[\langle\alpha\rangle\varphi],$$

We have

$$\dfrac{\dfrac{\dfrac{\sum' \quad \forall x..\mathscr{A}[\langle\alpha\rangle\varphi]}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}\,\forall\text{-EV}}{\mathscr{A}^a_x[\langle\alpha\rangle\varphi].}\,\equiv$$

Ad. (II): Since

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}^a_x[\tilde{\exists}\bar{y}.\varphi^a_x] \text{ and } \mathscr{A}[\langle\alpha\rangle\varphi]^a_x \equiv \mathscr{A}^a_x[\langle\alpha^a_x\rangle\varphi^a_x],$$

we have

$$\dfrac{\dfrac{\dfrac{\sum' \quad \forall x..\mathscr{A}[\langle\alpha\rangle\varphi]}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}\,\forall\text{-EV}}{\mathscr{A}^a_x[\langle\alpha^a_x\rangle\varphi^a_x].}\,\equiv$$

Ad. (III): By the condition (ix-a), $a \equiv x$; so

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]^a_x \equiv \mathscr{A}[\tilde{\exists}\bar{y}.\varphi] \quad \text{and} \quad \mathscr{A}[\langle\alpha\rangle\varphi]^a_x \equiv \mathscr{A}[\langle\alpha\rangle\varphi].$$

Hence we have

$$\dfrac{\dfrac{\dfrac{\sum'}{\forall x..\mathscr{A}[\langle\alpha\rangle\varphi]}}{\mathscr{A}[\langle\alpha\rangle\varphi]^{a}_{x}}\,\forall\text{-EV}}{\mathscr{A}[\langle\alpha\rangle\varphi].}\,\equiv$$

6.2.5. $\tilde{\exists}$-**I** Consider the following proof:

$$\dfrac{\sum}{\textbf{let } \bar{x} \leftarrow \bar{\tau} \textbf{ in } \tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi}{\tilde{\exists}\bar{x}\tilde{\exists}\bar{y}_1 \ldots \tilde{\exists}\bar{y}_m.\varphi.}$$

The argument in Section 5.3.6 is completely applicable in this case. We however treat the case where $\bar{\tau} \equiv \bar{x}$ separately, because this case produces simpler programs. From the induction hypothesis, we have

$$\dfrac{\sum'}{\textbf{let } \bar{x} \leftarrow \bar{x} \textbf{ in } \langle\alpha\rangle\varphi.}$$

Then we have

$$\dfrac{\dfrac{\sum'}{\textbf{let } \bar{x} \leftarrow \bar{x} \textbf{ in } \langle\alpha\rangle\varphi.}}{\langle \textbf{null}\colon (\bar{x};\, \bar{x});\, \alpha\rangle\varphi.}\,\langle\textbf{null}\rangle\textbf{and}\langle;\rangle$$

6.2.6. $\tilde{\exists}$-**E** Assume that the proof is of the following form:

$$\dfrac{\dfrac{\prod \quad \tilde{\exists}\bar{x}.\tilde{\exists}\bar{y}.\varphi}{\mathscr{A}[\tilde{\exists}\bar{z}.\psi]}\,\langle\bar{a}\rangle \qquad \dfrac{[(\tilde{\exists}\bar{y}.\varphi)^{\bar{a}}_{\bar{x}}]}{\sum\ \mathscr{A}[\tilde{\exists}\bar{z}.\psi]}}{\mathscr{A}[\tilde{\exists}\bar{z}.\psi]}$$

Let $\bar{x}'$ be the sequence of x_i's not occurring in $\bar{y}$ but occurring free in the assumptions above the major premise, and $\bar{x}''$ be the sequence of x_i's not occurring in $\bar{y}$ and not occurring free in the assumptions above the major premise. By the condition (x-a), $\bar{x}' \subseteq \mathbf{FV}(\tilde{\exists}\bar{y}.\varphi)$. So, from the induction hypothesis, we have

$$\prod' \quad \langle\alpha\colon (\bar{x}',\, \bar{x}'',\, \bar{y};\, \bar{x}',\, \bar{v})\rangle\varphi \qquad \text{and} \qquad \dfrac{\langle\xi\colon (\bar{y};\, \bar{a}',\, \bar{v})\rangle\varphi^{\bar{a}';\bar{a}''}_{\bar{x}',\bar{x}''}}{\sum'\ \mathscr{A}[\langle\beta\colon (\bar{z};\, \bar{w})\rangle\psi]}$$

where $\bar{a}'$ (resp. $\bar{a}''$) are a_is corresponding to $\bar{x}'$ (resp. $\bar{x}''$). By Lemma 5.4 we can augment the right variables of ξ in Σ' by fresh variables $\bar{b}$ compatible to $\bar{y}$. Let γ be $\textbf{null}\colon (;\, \bar{v}');\bar{y}\colon = \bar{b}$, where $\bar{v}' = \mathbf{RV}\,\xi$. Then γ has the same p-type as ξ. We

also define $\bar{u}$ as follows:

$$u_i \equiv \begin{cases} a_i & \text{if } x_i \in \bar{z}, \\ & \text{or } x_i \text{ occurs free in the assumptions above the minor premise,} \\ & \text{or } x_i \text{ is bound in } \mathscr{A}[*] \\ x_i & \text{otherwise,} \end{cases}$$

and let $\bar{u}'$ (resp. $\bar{u}''$) be u_i's corresponding to $\bar{x}'$ (resp. $\bar{x}''$). Then if $x_i \in \mathbf{LV}\beta \cup \mathbf{RV}\beta$, $u_i \equiv a_i$; so $\bar{u}$ are renamable to $\bar{a}$ in β; hence

$$\beta_\xi[\gamma] \equiv ((\beta^{\bar{u}',\bar{u}''}_{\bar{a}',\bar{a}''})_\xi[\gamma])^{\bar{a}',\bar{a}'',\bar{b}}_{\bar{u}',\bar{u}'',\bar{b}}.$$

Thus we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\prod'}{\langle\alpha\rangle\varphi}}{\langle\bar{u}':=\bar{x}';\alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}}\rangle\varphi^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}}}\ \text{lemma 5.10}(*) \qquad \dfrac{\sum'_\xi[\gamma]}{\mathscr{A}[\ \overline{\langle\beta_\xi[\gamma]\rangle\psi}\ ]}\equiv}{\mathscr{A}[\ \overline{\langle((\beta^{\bar{u}',\bar{u}''}_{\bar{a}',\bar{a}''})_\xi[\gamma])^{\bar{a}',\bar{a}'',\bar{b}}_{\bar{u}',\bar{u}'',\bar{b}}\rangle\psi}\ ]}\ \langle\bar{a}',\bar{a}'',\bar{b}\rangle\ \text{lemma 5.11}}{\mathscr{A}[\ \overline{\langle\bar{u}':=\bar{x}';\alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}}\rangle\langle(\beta^{\bar{u}',\bar{u}''}_{\bar{a}',\bar{a}''})_\xi[\gamma]\rangle\psi}\ ]}\ \langle;\rangle}{\mathscr{A}[\ \overline{\langle\bar{u}':=\bar{x}';\alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}},(\beta^{\bar{u}',\bar{u}''}_{\bar{a}',\bar{a}''})_\xi[\gamma]\rangle\psi}\ ]}\ \text{Lemma 5.12}(*)$$

$$\mathscr{A}[\ \overline{\langle\mathbf{local}\,\bar{u}',\bar{u}'',\bar{b}\,\mathbf{in}(\bar{u}':=\bar{x}';\alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}};(\beta^{\bar{u}',\bar{u}''}_{\bar{a}',\bar{a}''})_\xi[\gamma])\rangle\psi}\ ].$$

(*): When $\bar{a}' \equiv \bar{x}'$,

$$\langle\bar{u}':=\bar{x}';\ \alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}}\rangle\varphi^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}}$$

can be replaced by

$$\langle\alpha^{\bar{u}'',\bar{b}}_{\bar{x}'',\bar{y}}\rangle\varphi^{\bar{u}'',\bar{b}}_{\bar{x}'',\bar{y}}$$

and

$$\mathbf{local}\ \bar{u}',\ \bar{u}'',\ \bar{b}\ \mathbf{in}\ (\bar{u}':=\bar{x}';\ \alpha^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{x}',\bar{x}'',\bar{y}};\ (\beta^{\bar{u}',\bar{u}'',\bar{b}}_{\bar{a}',\bar{a}'',\bar{b}})_\xi[\gamma])$$

can be replaced by

$$\mathbf{local}\ \bar{u}'',\ \bar{b}\ \mathbf{in}\ (\alpha^{\bar{u}'',\bar{b}}_{\bar{x}'',\bar{y}};\ (\beta^{\bar{u}''}_{\bar{a}''})_\xi[\gamma]).$$

6.2.7. Ind This case is easily established just like 5.3.8 using the subcases used in **let**-I, so we omit the detailed proof. Here we only remark that if the conclusion of the inference is of the form $\forall\bar{x}.\ (\chi \supset \mathscr{A}[\ \overline{\exists\bar{y}_1 \ldots \exists\bar{y}_m.\varphi}\ ])$, then $\bar{x}$ do

not occur free in the assumptions by the condition (xi-b).

6.2.8. $\forall$-E_0 Consider the following proof:

$$(21)\qquad \frac{\prod\limits_{\triangle\bar{\tau}} \quad \sum\limits_{\forall\bar{x}.\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_{\bar{x}}[\bar{\tau}].}\,\forall\text{-}E_0$$

From the induction hypothesis, we have

$$\prod\limits'_{\triangle\bar{\tau}} \quad \text{and} \quad \sum\limits'_{\forall\bar{x}.\mathscr{A}[\langle\alpha\rangle\varphi]}.$$

Note that $x \in \mathbf{RV}\alpha$ whether x is bound in $\mathscr{A}[*]$ or not.

Now we consider the following three cases:

(I) x is bound in $\mathscr{A}[*]$;
(II) x is not bound in $\mathscr{A}[*]$, and $x \notin \bar{y}$;
(III) x is not bound in $\mathscr{A}[*]$, and $x \in \bar{y}$.

Ad. (I): We have

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[\tau] \equiv \mathscr{A}_x[\tau][\tilde{\exists}\bar{y}.\varphi].$$

Then by Lemma 5.19,

$$a = \tau \wedge \mathscr{A}[\langle\alpha\rangle\varphi]^a_x \rightarrow \mathscr{A}_x[\tau][\langle\alpha\rangle\varphi].$$

Hence

$$\frac{\dfrac{\prod\limits'_{\triangle\tau} \quad \sum\limits'_{\forall x.\mathscr{A}[\langle\alpha\rangle\varphi]}}{\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,A[\langle\alpha\rangle\varphi]}\,\forall\text{-E} \qquad \dfrac{\overset{\text{(let-E)}}{a=\tau} \quad \overset{\text{(let-E)}}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}}{\mathscr{A}_x[\tau][\langle\alpha\rangle\varphi]}\,\text{Lemma 5.19}}{\mathscr{A}_x[\tau][\langle\alpha\rangle\varphi].}\,\langle a\rangle\text{let-E}$$

Ad. (II): We have

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[\tau] \equiv \mathscr{A}_x[\tau][\tilde{\exists}\bar{y}.\varphi_x[\tau]].$$

Then by Lemma 5.19,

$$a = \tau \wedge \mathscr{A}[\langle\alpha\rangle\varphi]^a_x \rightarrow \mathscr{A}_x[\tau][\langle\mathbf{let}\ x \leftarrow \tau\ \mathbf{in}\ \alpha\rangle\varphi_x[\tau]].$$

Hence

$$\frac{\dfrac{\prod\limits'_{\triangle\tau} \quad \sum\limits'_{\forall x.\mathscr{A}[\langle\alpha\rangle\varphi]}}{\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,A[\langle\alpha\rangle\varphi]}\,\forall\text{-E} \qquad \dfrac{\overset{\text{(let-E)}}{a=\tau} \quad \overset{\text{(let-E)}}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}}{\mathscr{A}_x[\tau][\langle\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,\alpha\rangle\varphi_x[\tau]]}\,\text{Lemma 5.19}}{\mathscr{A}_x[\tau][\langle\mathbf{let}\,x\leftarrow\tau\,\mathbf{in}\,\alpha\rangle\varphi_x[\tau]].}\,\langle a\rangle\text{let-E}$$

Ad. (III): In this case,

$$\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[\tau] \equiv \mathscr{A}_x[\tau][\tilde{\exists}\bar{y}.\varphi].$$

From Proposition 3.6, the following lemma holds:

Lemma 6.4. *If $\bar{x}$ are marked variables which are not bound in $\mathscr{A}[*]$, $\bar{\tau}$ are substitutible for $\bar{x}$ in $\mathscr{A}[*]$, and $\bar{a}$ are fresh variables compatible with $\bar{x}$, then*

$$\vdash_{\mathrm{PLi}} \bar{a} = \bar{\tau} \wedge \mathscr{A}[\langle\alpha\rangle\varphi]^{\bar{a}}_{\bar{x}} \leftrightarrow \triangle\bar{\tau} \wedge \mathscr{A}_{\bar{x}}[\bar{\tau}][\langle \bar{x} := \bar{\tau};\ \alpha\rangle\varphi].$$

Using this lemma, we have

$$\dfrac{\dfrac{\prod'_{\triangle\tau} \quad \sum'_{\forall x.\mathscr{A}[\langle\alpha\rangle\varphi]}}{\mathbf{let}\, x \leftarrow \tau \,\mathbf{in}\, \mathscr{A}[\langle\alpha\rangle\varphi]}\,\forall\text{-E} \qquad \dfrac{\overset{(let-E)}{a=\tau} \quad \overset{(let-E)}{\mathscr{A}[\langle\alpha\rangle\varphi]^a_x}}{\mathscr{A}_x[\tau][\langle x:=\tau;\alpha\rangle\varphi]}\,\text{Lemma 6.4}}{\mathscr{A}_x[\tau][\langle x:=\tau;\alpha\rangle\varphi].}\,\langle a\rangle\text{let-E}$$

We note here that $x \in \mathbb{RV}\alpha$ but $x \notin \mathbb{RV}(x := \tau;\ \alpha)$ unless $x \in \mathbb{FV}\tau$. If x occurs free in the assumption of Π or Σ, then x must occur as a right variable of the finally obtained program. Hence, if the p-type of $x := \tau;\ \alpha$ is not the desirable one, we must augment x as a right variable as follows:

$$\dfrac{\mathscr{A}_x[\tau][\langle x:=\tau;\alpha\rangle\varphi]}{\mathscr{A}_x[\tau][\mathbf{null}:(;x);x:=\tau;\alpha].}\,\langle\mathbf{null}\rangle$$

6.2.9. $\tilde{\exists}$-$\mathbb{I}_0$ Consider the following proof:

$$\dfrac{\prod_{\triangle\bar{\tau}} \quad \sum_{(\tilde{\exists}\bar{y}.\varphi)_{\bar{x}}[\bar{\tau}]}}{\tilde{\exists}\bar{x}.\tilde{\exists}\bar{y}.\varphi.}\,\tilde{\exists}\text{-}\mathbb{I}_0$$

We note that $\bar{x}$ and $\bar{y}$ may be overlapped. Let $\bar{x}'$ be x_i's not in $\bar{y}$ and $\bar{\tau}'$ be τ_i's corresponding to $\bar{x}'$, then

$$(\tilde{\exists}\bar{y}.\varphi)_{\bar{x}}[\bar{\tau}] \equiv \tilde{\exists}\bar{y}.(\varphi_{\bar{x}'}[\bar{\tau}']).$$

So, from the induction hypothesis, we have

$$\prod'_{\triangle\bar{\tau}} \quad \text{and} \quad \sum'_{\langle\alpha:(\bar{y};\bar{u})\rangle\varphi_{\bar{x}'}[\bar{\tau}']}.$$

Hence we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\prod'}{\triangle\bar{\tau}}}{\triangle\bar{\tau}'} \quad \dfrac{\overset{\text{(let-I)}}{\bar{a}'=\bar{\tau}'} \quad \overset{(\langle\alpha\rangle\text{-rule})}{\varphi_{\bar{x}'}[\bar{\tau}']}}{\varphi^{\bar{a}'}_{\bar{x}'}}\text{Substitution Lemma}}{\mathbf{let}\,\bar{x}'\leftarrow\bar{\tau}'\,\mathbf{in}\,\varphi}\langle\bar{a}'\rangle\text{let-I}}{\langle\alpha:(\bar{y};\bar{u})\rangle\varphi_{\bar{x}'}[\bar{\tau}']\ \langle\bar{x}':=\bar{\tau}'\rangle\varphi}\langle:=\rangle\text{-I}}{\langle\alpha;\bar{x}':=\bar{\tau}'\rangle\varphi}\langle\bar{y}\rangle\langle\alpha\rangle\text{-rule}$$

where $\bar{a}'$ be fresh variables compatible with $\bar{x}'$.

In case $\bar{\tau}\equiv\bar{x}$, we can use **null**: $(\bar{x}';\ \bar{x}')$ instead of $\bar{x}':=\bar{\tau}'$.

Example 6. In Example 1, we can use z for the eigen variable b in $\sum$. This change does not violate the condition of Theorem 6.1. For subproofs corresponding to $\prod$, $\sum_1$ and $\sum_2$, $\prod'$ and $\sum'_1$ are the same as Example 4, but $\sum'_2$ becomes as follows:

$$\sum'_2\quad\dfrac{\dfrac{\dfrac{a\neq 0}{a-1+1=a}\text{PA} \quad \dfrac{F(a-1,\,z)}{F(a-1+1,(a-1+1)\cdot z)}\text{FACT}}{F(a,\,a\cdot z)}\text{Substitution Lemma}}{\langle z:=a\cdot z\rangle F(a,\,z).}$$

Then the proof Σ' corresponding to Σ becomes:

$$\dfrac{\begin{array}{cc} a\neq 0,\forall x.(x<a\supset\langle\xi\rangle F(x,\,z)) & \overset{(\langle\alpha\rangle\text{-rule})}{a\neq 0, F(a-1,\,z)} \\ \sum'_1 & \sum'_2 \\ \langle\mathbf{let}\,x\leftarrow a-1\,\mathbf{in}\,\xi:(z;\,x)\rangle F(a-1,\,z) & \langle z:=a\cdot z\rangle F(a,\,z)\end{array}}{\langle\mathbf{let}\,x\leftarrow a-1\,\mathbf{in}\,\xi:(z;\,x);\,z:=a\cdot z\rangle F(a,\,z).}\langle z\rangle,\langle\alpha\rangle\text{-rule},\langle;\rangle$$

From this $\sum'$ and $\prod'$, we can extract the desired program

$$\mu\xi:(z;x).(x=0\rightarrow z:=1\ \mid\ x\neq 0\rightarrow(\mathbf{let}\,x\leftarrow x-1\,\mathbf{in}\,\xi:(z;x));z:=x\cdot z)$$

just like Example 5.

6.3. Extracting while programs

Takasu [19, 20] showed that a while program can be extracted from a proof using templates of inference. Using **PLi**, his result can be applied to our framework. For example, a **while** statement can be extracted from the following form of inference.

Proposition 6.5. *Suppose*

$$\prod\vdash_{\tilde{\exists}\text{-UL}1}\Gamma,\ \bar{a}=\bar{\tau},\ \triangle\pi,\ \pi,\ \varphi\ \rightarrow\ \tilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\chi)\wedge\bar{\tau}\prec\bar{a}\wedge\triangle\pi\wedge\varphi)$$

where $\bar{a}$ are fresh variables and $\bar{a}$ and $\bar{x}$ are unmarked, then

$$\vdash_{\tilde{\exists}\text{-UL1}} \Gamma,\ \mathbf{TI}(\prec, \lambda\bar{x}.\chi),\ \mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi,\ \triangle\pi,\ \varphi \to \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi).$$

That is, the following derived rule holds:

$$\dfrac{\mathbf{TI}(\prec, \lambda\bar{x}.\chi) \quad \mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi \quad \triangle\pi \quad \varphi \quad \tilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi) \quad \begin{matrix}[\bar{a} = \bar{\tau},\ \triangle\pi,\ \pi, \varphi] \\ \vdots \end{matrix}}{\tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi).}\langle\bar{a}\rangle \tag{22}$$

Proof. By renaming variables, we may assume that $\bar{x} \cap (\mathbf{FV}\,\bar{\tau} \cup \mathbf{FV}\,\pi \cup \mathbf{FV}\,\varphi) = \phi$. Let

$$\Phi \triangleq \chi \wedge \bar{x} \prec \bar{a} \supset \forall\bar{y}.(\bar{x} = \bar{\tau} \wedge \Delta\pi \wedge \varphi \supset \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi))$$

$$IH \triangleq \forall\bar{x}.\Phi.$$

Then we have $\sum_1$ and $\sum_2$ such that

$$\sum\nolimits_1 \vdash_{\tilde{\exists}\text{-UL1}} \mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi,\ \bar{\tau} \prec \bar{a},\ \bar{b} = \bar{\tau},\ \triangle\pi,\ \Phi^{\bar{b}}_{\bar{x}} \to \varphi \supset \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi)$$

$$\sum\nolimits_2 \vdash_{\tilde{\exists}\text{-UL1}} \mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi,\ \triangle\pi,\ \forall\bar{x}.(\chi \supset (\forall\bar{y}.(\bar{x} = \bar{\tau} \wedge \triangle\pi \wedge \varphi \supset \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi)))) \to \varphi \supset \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi).$$

Now let Σ be

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi}(\tilde{\exists}\text{-E}) \quad \dfrac{}{\bar{\tau} \prec \bar{a}}(\tilde{\exists}\text{-E}) \quad \dfrac{}{\bar{b} = \bar{\tau}}(\text{let-E}) \quad \dfrac{}{\triangle\pi}(\tilde{\exists}\text{-E}) \quad \dfrac{\dfrac{}{\forall\bar{x}.\Phi}IH}{\Phi^{\bar{b}}_{\bar{x}}}\forall\text{-EV}}{\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\Phi}\forall\text{-E} \quad \dfrac{\dfrac{}{\varphi}(\tilde{\exists}\text{-E}) \quad \dfrac{}{\sum_1}}{\varphi \supset \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi)}\supset\text{-E}}{\tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi)}\langle\bar{b}\rangle\text{let-E}}{\bar{a} = \bar{\tau}, \triangle\pi, \pi, \varphi} \quad \begin{matrix}\vdots \\ \vdots\end{matrix}}{\tilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi) \quad \tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi)}\langle\bar{y}\rangle\tilde{\exists}\text{E}$$
$$\tilde{\exists}\bar{y}.(\varphi \wedge \neg \pi).$$

Then (22) is derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\pi \vee \neg\pi}\quad \dfrac{\dfrac{\dfrac{\dfrac{}{\bar{a}=\bar{\tau},\ \triangle\pi,\ \pi,\ \varphi,\ IH}\ (\supset\text{-I})\ (\supset\text{-I})\ (\vee\text{-E})\ (\supset\text{-I})\ (\text{IND})}{\sum}}{\widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi)}\ (\supset\text{-I})}{\triangle\pi}\ (\supset\text{-I})}{\bar{a}=\bar{\tau} \wedge \triangle\pi \wedge \varphi \supset \widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi)}\ \langle\bar{y}\rangle\ \forall\text{-I}}{\mathrm{TI}(\prec, \lambda\bar{x}.\chi)\ \forall\bar{y}.(\bar{a}=\bar{\tau} \wedge \triangle\pi \wedge \varphi \supset \widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi))}\ \langle\bar{a}\rangle\ \text{Ind}}$$

$$\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi,\ \triangle\pi,\ \forall\bar{x}.(\chi \supset (\forall\bar{y}.(\bar{x}=\bar{\tau} \wedge \triangle\pi \wedge \varphi \supset \widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi))))$$

$$\dfrac{\varphi \qquad \dfrac{\sum_2}{\varphi \supset \widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi)}}{\widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi)}\ \supset\text{-E}$$

∎

This proof satisfies $\widetilde{\exists}$-**wCV** and the conditions of Theorem 6.1. So we can apply the theorem to this proof, then (22) becomes the following derived rule in **PLi**:

$$\dfrac{\begin{array}{c}[\bar{a}=\bar{\tau},\ \triangle\pi,\ \pi,\ \varphi]\\ \vdots\end{array}}{\mathbf{TI}(\prec, \lambda\bar{x}.\chi)\ \mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi\ \ \triangle\pi\ \ \varphi\ \ \langle\alpha\rangle((\mathbf{let}\,\bar{x} \leftarrow \widetilde{\tau}\,\mathbf{in}\,\chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi)}\langle\bar{a}\rangle$$
$$\langle\mu\xi.(\pi \to \alpha \mid \neg\pi \to \mathbf{null})\rangle(\varphi \wedge \neg\pi)$$

where α is the program obtained from $\prod$. Thus we obtained the desired while program:

$$\mathbf{while}\ \pi\ \mathrm{do}\ \alpha.$$

This derived rule just corresponds to {**while**} rule in **tHL**.

We usually use Proposition 6.5 in the following form:

Corollary 6.6. (While Rule) *Suppose*

(i) $\prod \vdash_{\widetilde{\exists}\text{-UL}1} \Gamma \to \widetilde{\exists}\bar{y}.((\mathbf{let}\ \bar{x} \leftarrow \bar{\tau}\ \mathbf{in}\ \chi) \wedge \triangle\pi \wedge \varphi)$,

(ii) $\sum \vdash_{\widetilde{\exists}\text{-UL}1} \Gamma', \bar{a}=\bar{\tau}, \triangle\pi, \pi, \varphi \to \widetilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi)$

where $\bar{a}$ are fresh variables and $\bar{a}$ and $\bar{x}$ are unmarked, then

$$\vdash_{\widetilde{\exists}\text{-UL}1} \Gamma, \Gamma', \mathbf{TI}(\prec, \lambda\,\bar{x}.\,\chi) \to \widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi).$$

That is, the following derived rule holds:

$$\dfrac{\begin{array}{c}[\bar{a}=\bar{\tau},\ \triangle\pi,\ \pi,\ \varphi]\\ \vdots\end{array}}{\mathbf{TI}(\prec, \lambda\bar{x}.\chi)\ \widetilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x} \leftarrow \bar{\tau}\,\mathbf{in}\,\chi) \wedge \triangle\pi \wedge \varphi)\ \widetilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x} \leftarrow \widetilde{\tau}\,\mathbf{in}\,\chi) \wedge \bar{\tau} \prec \bar{a} \wedge \triangle\pi \wedge \varphi)}\langle\bar{a}\rangle \tag{23}$$
$$\widetilde{\exists}\bar{y}.(\varphi \wedge \neg\pi).$$

Proof.

$$\dfrac{\dfrac{\overset{\bar{a}=\bar{\tau},\ \triangle\pi,\ \pi,\ \varphi}{\sum}}{\mathrm{TI}(\prec,\ \lambda\bar{x}.\chi)\ (\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\chi)\wedge\triangle\pi\wedge\varphi\ \ \tilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\chi)\wedge\bar{\tau}\prec\bar{a}\wedge\triangle\pi\wedge\varphi)}\langle\bar{a}\rangle\text{Prop. 6.5}}{\dfrac{\overset{\prod}{\tilde{\exists}\bar{y}.((\mathbf{let}\,\bar{x}\leftarrow\bar{\tau}\,\mathbf{in}\,\chi)\wedge\triangle\pi\wedge\varphi)}\qquad\tilde{\exists}\bar{y}.(\varphi\wedge\neg\pi)}{\tilde{\exists}\bar{y}.(\varphi\wedge\neg\pi)}\langle\bar{y}\rangle\tilde{\exists}\mathrm{E}}$$

From this proof, if α and β are programs obtained from $\prod$ and $\sum$, respectively, then, by Theorem 6.1, we have the following program:

$$\alpha;\ \textbf{while}\ \pi\ \text{do}\ \beta.$$

Example 7. (Integer square root program) The specification for a square root program on integers is described as follows:

$$Goal \triangleq \forall x.(0 \leq x \supset \tilde{\exists} y.\, Sqrt(x,\ y))$$

where $Sqrt(x, y)$ is a predicate expressing y is the square root of x, that is y is the largest number satisfying $y^2 \leq x$, as follows:

$$Sqrt(x,\ y) \triangleq 0 \leq y \wedge y^2 \leq x \wedge \forall z.(z^2 \leq x \supset z \leq y).$$

First we let P and Q be as follows:

$$P \triangleq y^2 \leq x \wedge 0 \leq y < z$$
$$Q \triangleq P \wedge x < z^2.$$

Suppose we are given

$$\sum_{Q1} \vdash 0 \leq x \ \rightarrow\ \tilde{\exists} y,\ z.(0 \leq z - y \wedge Q)$$
$$\sum_{Q2} \vdash a = z - y \wedge y + 1 \neq z \wedge Q \ \rightarrow\ \tilde{\exists} y,\ z.(0 \leq z - y < a \wedge Q)$$
$$\sum_{Q3} \vdash y + 1 = z \wedge Q \ \rightarrow\ Sqrt(x,\ y),$$

then we can construct a proof for *Goal* as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{0\leq x}{\sum_{Q1}}\qquad\dfrac{\overset{(1)}{a=z-y},\ \overset{(1)}{y+1\neq z},\ \overset{(1)}{Q}}{\sum_{Q2}}}{\tilde{\exists}y,z.(0\leq z-y\wedge Q)\quad\tilde{\exists}y,z.(0\leq z-y<a\wedge Q)}\langle a\rangle(1)\text{While rule}}{\tilde{\exists}y,z.(\neg(y+1\neq z)\wedge Q)}}{\tilde{\exists}y,z.(y+1=z\wedge Q)}}{\sum'_{Q3}}$$
$$\tilde{\exists}y.Sqrt(x,\ y)$$

where $\sum'_{Q3}$ is

$$\dfrac{\dfrac{\tilde{\exists}y,z.(y+1=z\wedge Q)}{\tilde{\exists}y.Sqrt(x,\ y).}\ \dfrac{\dfrac{\overset{(\tilde{\exists}E)}{y+1=z\wedge Q}}{\sum_{Q3}}\ \dfrac{Sqrt(x,\ y)}{\tilde{\exists}y.Sqrt(x,\ y)}\ \tilde{\exists}I_0}{\tilde{\exists}y.Sqrt(x,\ y).}\ \langle y,z\rangle\tilde{\exists}E$$

Hence, when we obtain programs α and β from $\sum_{Q1}$ and $\sum_{Q2}$, respectively, we have the following program satisfying the specification *Goal*:

local z in
(α; **while** $y+1\neq z$ **do** β).

Now we establish $\sum_{Q1}$, $\sum_{Q2}$ and $\sum_{Q3}$.

Ad. $\sum_{Q1}$: Since

$$0\leq x\ \rightarrow\ (0\leq 2\cdot(x+1)-z\wedge P)_{y,z}[0,\ 1],$$

we have

$$\sum_{P1}\vdash 0\leq x\ \rightarrow\ \tilde{\exists}y,\ z.(0\leq 2\cdot(x+1)-z\wedge P).$$

Since

$$a=2\cdot(x+1)-z\wedge z\leq x\wedge P\ \rightarrow\ (0\leq 2\cdot(x+1)-z<a\wedge P)_{y,z}[y,\ 2\cdot z],$$

we have

$$\sum_{P2}\vdash a=2\cdot(x+1)-z\wedge z\leq x\wedge P\ \rightarrow\ \tilde{\exists}y,\ z.(0\leq 2\cdot(x+1)-z<a\wedge P).$$

Hence by While rule with $\sum_{P1}$ and $\sum_{P2}$,

$$0\leq x\ \rightarrow\ \tilde{\exists}y,\ z.(P\wedge\neg(z\leq x)).$$

Moreover we have

$$\vdash_{\text{ULI}} P\wedge\neg(z\leq x)\ \rightarrow\ 0\leq z-y\wedge Q.$$

Thus $\sum_{Q1}$ is obtained. The program corresponding to $\sum_{Q1}$ is

$$y,\ z:=0,\ 1;\ \textbf{while}\ z\leq x\ \textbf{do}\ z:=2\cdot z.$$

Ad. $\sum_{Q2}$: We use an auxiliary predicate R:

$$R\triangleq a=z-y\wedge y+1\neq z\wedge Q\wedge y<w<z.$$

We have

$$\sum_{R1}\vdash a=z-y\wedge y+1\neq z\wedge Q\ \rightarrow\ R_w[\tfrac{y+z}{2}].$$

Assume R and consider separate cases according to $w^2 \leq x \wedge x < w^2$, then we have

$$w^2 \leq x \to (0 \leq z - y < a \wedge Q)_y[w]$$
$$x < w^2 \to (0 \leq z - y < a \wedge Q)_z[w].$$

Hence we have

$$\sum_{R2} \vdash \forall w.(R \supset \tilde{\exists} y, z.(0 \leq z - y < a \wedge Q)).$$

Then $\sum_{Q2}$ can be constructed as follows:

$$\dfrac{\dfrac{a = z - y \wedge y + 1 \neq z \wedge Q \qquad \sum_{R1}}{R_w[\frac{y+z}{2}]} \qquad \dfrac{\sum_{R2}}{\dfrac{\forall w.(R \supset \tilde{\exists} y,z.(0 \leq z - y < a \wedge Q))}{R_w[\frac{y+z}{2}] \supset \tilde{\exists} y,z.(0 \leq z - y < a \wedge Q)}\ \forall\text{-E}_0}}{\tilde{\exists} y,z.(0 \leq z - y < a \wedge Q).}\ \supset\text{-E}$$

Since $\forall$-E_0 and $\vee$-E are used, the corresponding program is

$$\textbf{let } w \leftarrow \tfrac{y+z}{2} \textbf{ in}$$
$$(w^2 \leq x \to y := w \mid x < w^2 \to z := w).$$

Ad. $\sum_{Q3}$: This case is straightforward.
So we finally obtained the following program:

$$\begin{array}{l}\textbf{local } z \textbf{ in } (\\ \quad y,\ z := 0,\ 1;\ \textbf{while } z \leq x \textbf{ do } z := 2 \cdot z;\\ \quad \textbf{while } y + 1 \neq z \textbf{ do}\\ \quad\quad \textbf{let } w \leftarrow \tfrac{y+z}{2} \textbf{ in}\\ \quad\quad\quad (w^2 \leq x \to y := w \mid x < w^2 \to z := w)).\end{array}$$

6.4. Mathematical induction

Our system is not based on the usual mathematical induction but on the transfinite induction. We think the mathematical induction does not specify the algorithm explicitly. To a proof using the mathematical induction, we may associate two types of programs

(i) a program using counting down recursion;

(ii) a program using counting up loop.

Since we often use the mathematical induction rather than the transfinite induction, it is useful to examine how a program is extracted from a proof using the mathematical induction.

We now consider the following proof using the mathematical induction.

$$(24)\qquad \frac{\prod \quad \overset{[\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a]]}{\sum}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[0] \quad \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a+1]}\ \langle a\rangle \text{ mathematical induction}$$
$$\forall x.\mathscr{A}[\tilde{\exists}\bar{y}.\varphi].$$

6.4.1. Counting down program From (24), we can obtain the following proof using the transfinite induction,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a=0\vee a\neq 0}\quad \dfrac{\overset{(\vee\text{-E})}{a=0}\quad \overset{\prod}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[0]}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a]}\quad \dfrac{\overset{\sum_a[a-1]}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a-1+1]}\quad \dfrac{\overset{(\vee\text{-E})}{a\neq 0}}{a-1+1=a}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a]}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a]}\ \vee\text{-E}}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a]}\ \langle a\rangle \text{ IND}}{\forall x.\mathscr{A}[\tilde{\exists}\bar{y}.\varphi].}$$

$$\dfrac{\dfrac{\overset{(\vee\text{-E})}{a\neq 0}\qquad \overset{(\text{IND})}{\forall x.(x<a\supset \mathscr{A}[\tilde{\exists}\bar{y}.\varphi])}}{a-1<a}\ \forall\text{-E}_0\qquad \dfrac{a-1<a\supset \mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a-1]}{\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]_x[a-1]}\ \supset\text{-E}}{}$$

Now we transform this proof into a proof in **PLi**. From $\prod$ part, we have

$$(26)\qquad \mathscr{A}_x[0][\langle\alpha\rangle\varphi_x[0]]$$

for some program α, and from $\sum_a[a-1]$, we have

$$(27)\qquad \frac{\mathscr{A}_x[a-1][\langle\eta\rangle(\varphi_x[a-1])]}{\mathscr{A}_x[a][\langle\beta\rangle\varphi_x[a]]}\ \sum'$$

for some program $\beta[\eta]$. Let γ be

$$(x=0\to\alpha \mid x\neq 0\to\beta_n[\textbf{let } x\leftarrow x-1 \textbf{ in } \xi])$$

and the induction hypothesis be

$$\forall x.(x<a\supset\mathscr{A}[\langle\mu\xi.\gamma\rangle\varphi]).$$

From $\forall$-E$_0$ and $\supset$-E parts in (25),

$$\mathscr{A}_x[a-1][\langle\textbf{let } x\leftarrow a-1 \textbf{ in } \mu\xi.\gamma\rangle\varphi_x[a-1]].$$

Then from (27),

$$\mathscr{A}_x[a][\langle\beta_\eta[\textbf{let } x\leftarrow a-1 \textbf{ in } \mu\xi.\gamma]\rangle\varphi_x[a]].$$

From this, (26) and $\vee$-E,

$$\mathscr{A}_x[a][\langle(a=0\to\alpha \mid a\neq 0\to\beta_\eta[\textbf{let}\, x\leftarrow a-1\ \textbf{in}\ \mu\xi.\gamma])\rangle\varphi_x[a]].$$

By $\langle\mu\rangle$ axiom,

$$\mathscr{A}_x[a][\langle(\mu\xi^a_x.\gamma^a_x)\rangle\varphi_x[a]].$$

Hence by induction

$$\forall x.\,\mathscr{A}[\langle\mu\xi.\gamma\rangle\varphi].$$

6.4.2. Counting up program If $\mathscr{A}[\tilde{\exists}\bar{y}.\varphi]$ in (25) is simply of the form $\psi\supset\tilde{\exists}\bar{y}.\varphi$ such that x and $\bar{y}$ do not occur free in ψ, we can extract a while program, which computes values of $\bar{y}$ satisfying $\varphi_x[i]$ from $i=0$ to x successively.

We use Corollary 6.6 (While rule) to extract a **while** program. Let τ be $x-i$, π be $i<x$, $\prec$ be $<$ and χ be $\top$. We now check the condition (i) and (ii).

Ad. (i): We show

$$\Gamma,\ \psi\to\tilde{\exists}i,\ \bar{y}.\,\varphi_x[i].$$

This is obtained as follows:

$$\dfrac{\dfrac{\prod}{\psi\quad \psi\supset\tilde{\exists}\bar{y}.\varphi_x[0]}\supset\text{-E}\qquad \dfrac{\varphi_x[0]}{\tilde{\exists}i,\tilde{y}.\varphi_x[i]}\tilde{\exists}\text{I}_0}{\tilde{\exists}i,\bar{y}.\varphi_x[i].}\langle\bar{y}\rangle\tilde{\exists}\text{E}\qquad (\tilde{\exists}\text{-E})$$

From this, we obtain a program

$$a\ ;i:=0$$

where α is a program extracted from Π.

Ad. (ii): it is sufficient to show the following:

$$\Gamma',\ \psi,\ a=x-i,\ i<x,\ \varphi_x[i]\to\tilde{\exists}i,\ \bar{y}.(x-i<a\wedge\varphi_x[i]).$$

This is obtained as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi_x[i]}{\tilde{\exists}\bar{y}.\varphi_x[i]}\tilde{\exists}\text{I}_0}{\psi\supset\tilde{\exists}\bar{y}.\varphi_x[i]}\supset\text{-I}}{\dfrac{\psi\quad \psi\supset\tilde{\exists}\bar{y}.\varphi_x[i+1]}{\tilde{\exists}\bar{y}.\varphi_x[i+1]}\supset\text{-E}}\qquad \dfrac{\dfrac{a=x-i\quad i<x}{x-(i+1)<a}\qquad \varphi_x[i+1]}{\tilde{\exists}i,\bar{y}.(x-i<a\wedge\varphi_x[i])}(\tilde{\exists}\text{-E})}{\tilde{\exists}i,\bar{y}.(x-i<a\wedge\varphi_x[i]).}\langle\bar{y}\rangle\tilde{\exists}\text{E}$$

From this we can obtain

$$\beta_\xi[\textbf{null}:(\bar{y};\ \bar{y})];\ i:=i+1$$

where $\beta[\xi]$ is a program extracted from $\sum_a[i]$.

Then, by While rule, we obtain

$$\Gamma, \Gamma', \psi \longrightarrow \tilde{\exists} i, \bar{y}.(\varphi_x[i] \wedge i = x) \tag{28}$$

and a program

$$\alpha;\ i := 0;\ \textbf{while}\ i < x\ \text{do}\ (\beta_\xi[\textbf{null}:(\bar{y};\ \bar{y})];\ i := i + 1).$$

Hence we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\psi}(\supset\text{-I})}{\tilde{\exists} i,\bar{y}.(\varphi_x[i] \wedge i = x)}\ \text{using(28)} \qquad \dfrac{\dfrac{}{\varphi_x[i]}(\tilde{\exists}\text{E}) \quad \dfrac{}{i = x}(\tilde{\exists}\text{E})}{\varphi}}{\tilde{\exists}\bar{y}.\varphi}\ \tilde{\exists}\text{I}_0}{\tilde{\exists}\bar{y}.\varphi}\ \langle i,\bar{y}\rangle\tilde{\exists}\text{E}}{\psi \supset \tilde{\exists}\bar{y}.\varphi}\ \supset\text{-I}}{\forall x.(\psi \supset \tilde{\exists}\bar{y}.\varphi)}\ \langle x\rangle\forall\text{-I}$$

and finally we obtain

local i **in** $(\alpha;\ i := 0;$
while $i < x$ do $(\beta_\xi[\textbf{null}:(\bar{y};\ \bar{y})];\ i := i + 1))$.

6.5. Or elimination

In our method, main drawback is found on the restriction of $\vee$-E rule. We can only use boolean expressions as major premises if the minor premise has $\tilde{\exists}$ quantifiers. If we prove a specification using a case analysis by *prime*$(x) \vee \neg$ *prime*(x) but *prime* is not executable predicate symbol, we must transform this proof into another one in which the major premise of the $\vee$-E is a boolean expression.

In this section, we briefly sketch how to convert such a proof. Now consider the following:

$$\dfrac{\sum \quad \prod_1^{[\psi_1]} \quad \cdots}{\vee_i \psi_i \quad \mathscr{A}[\tilde{\exists}\bar{x}.\varphi] \quad \cdots \qquad \mathscr{A}[\tilde{\exists}\bar{x}.\varphi]} \tag{29}$$

where each ψ_i is $\tilde{\exists}$-free.

It is already well-known that $\vee$ connective can be deleted using other logical connective if, for example, natural numbers are defined. We here assume the boolean sort which consists of two elements, **true** and **false**. Then we can replace $\vee_i \psi_i$ in $\sum$ by

$$\tilde{\exists}\bar{b}.((\vee_i(b_i = \textbf{true})) \wedge \wedge_i(b_i = \textbf{true} \supset \psi_i)) \tag{30}$$

because the following proposition holds.

Proposition 6.7.

$$\vdash \vee_i \psi_i \leftrightarrow \tilde{\exists}\bar{b}.((\vee_i(b_i = \mathbf{true})) \wedge \wedge_i(b_i = \mathbf{true} \supset \psi_i)).$$

The following corollary corresponds to $\vee$-I and $\vee$-E;

Corollary 6.8. *The following derived rules hold:*

$$\frac{\psi_k}{\tilde{\exists}\bar{b}.((\vee_i(b_i = \mathbf{true})) \wedge \wedge_i(b_i = \mathbf{true} \supset \psi_i))}$$

and

$$\frac{\tilde{\exists}\bar{b}.((\vee_i(b_i = \mathbf{true})) \wedge \wedge_i(b_i = \mathbf{true} \supset \psi_i)) \quad \varphi \quad \begin{matrix}[\psi_1] \\ \vdots \\ \cdots\end{matrix}}{\varphi}$$

Let $\Phi \triangleq (\vee_i(b_i = \mathbf{true})) \wedge \wedge_i(b_i = \mathbf{true} \supset \psi_i)$. If $\vee_i\psi_i$ does not occur as a subformula in any assumption of $\sum$ nor any instance of the axiom EM used in $\sum$, then we replace all the occurrences of $\vee_i\psi_i$ by Φ in $\sum$ with suitable renaming of variables using the above corollary. Let $\sum'$ be the proof of Φ obtained by this replacement. Then we can construct the following proof from (29), in which $\vee_i\psi_i$ does not occur as the major premise of $\vee$ -E.

$$\dfrac{\sum' \quad \dfrac{\overset{(\tilde{\exists}\text{-E})}{\Phi}}{\vee_i(b_i = \mathbf{true})}\wedge\text{-E} \quad \dfrac{\overset{(\tilde{\exists}\text{-E})}{\Phi}}{\wedge_i\triangle(b_i = \mathbf{true})} \quad \dfrac{\dfrac{\dfrac{\overset{(\vee\text{-E})}{b_1 = \mathbf{true}} \quad \dfrac{\wedge_i(b_i = \mathbf{true} \supset \psi_i)}{b_1 = \mathbf{true}_1 \supset \psi_1}\vee\text{-E}}{\psi_1}\supset\text{-E}}{\prod_1} \quad \cdots}{\mathscr{A}[\tilde{\exists}\bar{x}.\varphi]}(\vee\text{-E})}{\mathscr{A}[\tilde{\exists}\bar{x}.\varphi]}\langle\bar{b}'\rangle\tilde{\exists}\text{E}}{\mathscr{A}[\tilde{\exists}\bar{x}.\varphi].}$$

If we introduce a sort for n-element sets, say $\mathbb{S}_n$ then we can use only one variable for $\bar{b}$ in (30):

$$\tilde{\exists}b{:}\mathbb{S}_n.\ \wedge_i(b = \mathbb{c}_j \supset \psi_i)$$

where $\mathbb{c}_i$ is the i-th element of $\mathbb{S}_n$. In particular, when $n = 2$, (30) is simplified:

$$\tilde{\exists}b{:}boolean.\ ((b = \mathbf{true} \supset \psi_1) \wedge (b = \mathbf{false} \supset \psi_2)).$$

Example 8. (Prime predicate) We shall demonstrate how a proof of

(31) $$2 \leq x \rightarrow prime(x) \wedge \neg prime(x)$$

is transformed when we replace

$$prime(x) \wedge \neg prime(x)$$

by

$$\exists b.\ ((b = \textbf{true} \supset prime(x)) \wedge (b = \textbf{false} \supset \neg prime(x))).$$

Suppose *prime* is defined as follows:

$$P(u, v) \triangleq \forall z.\ (2 \leq z < v \supset \neg div(z, u))$$

$$prime(u) \triangleq P(u, u)$$

where $div(z, u)$ is the predicate representing u is *divisible* by z. Then the following hold:

P1 $\vdash P(x, 2)$

P2 $\vdash \neg div(a, x),\ P(x, a-1) \rightarrow P(x, a)$

P3. $\vdash div(a, x) \rightarrow \neg P(x, a)$

P4 $\vdash \neg P(x, a) \rightarrow \neg P(x, a+1)$.

Let IH be

$$IH \triangleq \forall y.\ (2 \leq y \wedge y < a \supset P(x, y) \vee \neg P(x, y)).$$

Now we consider the following proof of (31)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{TI}(<, \lambda w.(2 \leq w))}}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{2 \leq a}\,(\text{IND})}{a = \vee\ 2 < a}}{P(x, a) \vee \neg P(x, a)}\,\vee\text{-I} \qquad \dfrac{\dfrac{}{a = 2}\,(\vee\text{-E})}{P(x, a)}\,P1}{P(x, a) \vee \neg P(x, a)}\,\vee\text{-E}}{\forall y.(2 \leq y \supset P(x, y) \vee \neg P(x, y))}\,\langle a \rangle, \text{IND}}}{2 \leq x \supset P(x, x) \vee \neg P(x, x)}\,\forall\text{-E}_0 \qquad 2 \leq x}{P(x, x) \vee \neg P(x, x)}\,\supset\text{-E}}{prime(x) \vee \neg prime(x)}\,\equiv$$

$$\dfrac{\dfrac{}{2 < a,\ IH}\,(\vee\text{-E})\ (\text{IND}) \qquad \sum}{P(x, a) \vee \neg P(x, a)}$$

where $\sum$ is

$$\dfrac{\dfrac{\dfrac{}{2 < a,\ IH}}{\sum_0} \qquad \dfrac{\dfrac{}{2 < a,\ P(x, a-1)}\,(\vee\text{-E})}{\sum_1} \qquad \dfrac{\dfrac{}{2 < a,\ \neg P(x, a-1)}\,(\vee\text{-E})}{\sum_2}}{P(x, a) \vee \neg P(x, a)}\,\vee\text{-E}$$

and $\sum_0$, $\sum_1$ and $\sum_2$ are

$\sum_0$

$$\dfrac{\sum_0 \quad \dfrac{2<a}{2\leq a-1 \wedge a-1<a} \qquad \dfrac{\overset{IH}{\forall y.(2\leq y \wedge y<a \supset P(x,y) \vee \neg P(x,y))}}{2\leq a-1 \wedge a-1<a \supset P(x,a-1) \vee \neg P(x,a-1)}\,\forall\text{-E}_0}{P(x,a-1) \vee \neg P(x,a-1)}\,\supset\text{-E}$$

$$\dfrac{\sum_1 \quad \prod \quad \dfrac{\dfrac{\overset{(\vee\text{-E})}{div(a,x)}}{\neg P(x,a)}\,\text{P2}}{P(x,a) \vee \neg P(x,a)}\,\vee\text{-I} \qquad \dfrac{\dfrac{\overset{(\vee\text{-E})}{2<a \;\; \neg div(a,x) \;\; P(x,a-1)}}{P(x,a)}\,\text{P3}}{P(x,a) \vee \neg P(x,a)}\,\vee\text{-I}}{P(x,a) \vee \neg P(x,a)}\,\vee\text{-E}$$

$$\sum_2 \qquad \dfrac{\dfrac{2<a \;\; \neg P(x,a-1)}{\neg P(x,a)}\,\text{P4}}{P(x,a) \vee \neg P(x,a).}\,\vee\text{-I}$$

Next we transform these proofs. We introduce several abbreviations:

$$T(b) \triangleq b = \textbf{true}$$
$$F(b) \triangleq b = \textbf{false}$$
$$Q(b,u,v) \triangleq (T(b) \supset P(u,v)) \wedge (F(b) \supset \neg P(u,v))$$

where b is a variable of sort *boolean*. Then the following hold:

Q1 $\vdash T(b),\ Q(b,x,a-1) \rightarrow P(x,a-1)$
Q2 $\vdash F(b),\ Q(b,x,a-1) \rightarrow \neg P(x,a-1).$

The sequents corresponding to Corollary 6.8 are:

R1 $\vdash P(x,a) \rightarrow \exists b.\ Q(b,x,a)$
R2 $\vdash \neg P(x,a) \rightarrow \exists b.\ Q(b,x,a).$

R1 and R2 correspond to $\wedge$-I rules and these produce the following program:

R1: $b := \textbf{true}$
R2: $b := \textbf{false}.$

Let IH' be

$$IH' \triangleq \forall y.(2 \leq y \wedge y < a \supset \exists b.\ Q(b,x,y)),$$

then the proof for (31) is transformed as follows:

$$
(32)\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{TI}(<,\lambda w.(2\le w))}{\forall y.(2\le y\supset \tilde{\exists} b.Q(b,x,y))}\,\forall\text{-E}_0}{2\le x\supset \tilde{\exists} b.Q(b,x,x)}\,\supset\text{-E}}{\tilde{\exists} b.Q(b,x,x)}\,\equiv}{\tilde{\exists} b.((T(b)\supset prime(x))\wedge(F(b)\supset\neg prime(x)))}
$$

where Σ' is

$$
\dfrac{\dfrac{2<a,\ IH'}{\sum_0'}\qquad \dfrac{\dfrac{}{T(b)\vee F(b)}\qquad \dfrac{\overset{(\vee\text{-E})}{\overset{(\tilde{\exists}\text{-E})}{\dfrac{T(b)\quad Q(b,x,a-1)}{2<a,\quad P(x,a-1)}\,Q1}}\ \sum_1'\ \tilde{\exists} b.Q(b,x,a)\qquad \overset{(\vee\text{-E})}{\overset{(\tilde{\exists}\text{-E})}{\dfrac{F(b)\quad Q(b,x,a-1)}{2<a,\quad \neg P(x,a-1)}\,Q2}}\ \sum_2'\ \tilde{\exists} b.Q(b,x,a)}{\tilde{\exists} b.Q(b,x,a)}\,\vee\text{-E}}{\tilde{\exists} b.Q(b,x,a-1)}\,\tilde{\exists}\text{-E}}{\tilde{\exists} b.Q(b,x,a)}
$$

where $\sum_0'$, $\sum_1'$ and $\sum_2'$ are:

$$
\sum_0'\quad \dfrac{\dfrac{2<a}{2\le a-1\wedge a-1<a}\qquad \dfrac{IH'}{\dfrac{\forall y.(2\le y\wedge y<a\supset \tilde{\exists} b.Q(b,x,y))}{2\le a-1\wedge a-1<a\supset \tilde{\exists} b.Q(b,x,a-1)}\,\forall\text{-E}_0}\,\supset\text{-E}}{\tilde{\exists} b.Q(b,x,a-1)}
$$

$$
\sum_1'\quad \dfrac{\dfrac{\prod}{div(a,x)\vee\neg div(a,x)}\qquad \dfrac{\overset{(\vee\text{-E})}{\dfrac{div(a,x)}{\neg P(x,a)}\,P2}}{\tilde{\exists} b.Q(b,x,a)}\,R2\qquad \dfrac{\dfrac{T(b)\quad Q(b,x,a-1)}{2<a\quad \neg div(a,x)\qquad P(x,a-1)}\,Q1,P3}{P(x,a)}\,R1}{\tilde{\exists} b.Q(b,x,a)}\,\tilde{\exists}\text{-E}
$$

$$
\sum_2'\quad \dfrac{\dfrac{\dfrac{F(b)\quad Q(b,x,a-1)}{2<a\quad \neg P(x,a-1)}\,Q2}{\neg P(x,a)}\,P4}{\tilde{\exists} b.Q(b,x,a).}\,R2
$$

Now we extract programs from these proof fragments. First we use a program variable ξ:$(b; x, y)$ for $\tilde{\exists}b.\ in\ IH'$. Then we can extract the following programs from $\sum'_0$, $\sum'_1$ and $\sum'_2$:

$$\begin{aligned}&\textbf{let } y \leftarrow a - 1 \textbf{ in } \xi,\\ &(div(a, x) \rightarrow b := \textbf{false} \mid \neg\, div(a, x) \rightarrow b := \textbf{true})\\ \text{and } &b := \textbf{false}.\end{aligned}$$

Moreover we can *optimize* $\sum'_2$ as follows:

$$\dfrac{\dfrac{\dfrac{F(b)\quad Q(b, x, a-1)}{\quad}Q2 \qquad 2<a \qquad \neg P(x, a-1)}{\quad}P4}{\dfrac{F(b) \qquad \neg P(x, a)}{\dfrac{Q(b, x, a)}{\tilde{\exists}b.Q(b, x, a).}\tilde{\exists}I}}$$

From this, we obtain the following simple program:

$$\textbf{null}:(b; b).$$

Then from $\sum'$, we have

$$\begin{aligned}&\textbf{let } y \leftarrow a - 1 \textbf{ in } \xi;\\ &(b = \textbf{true} \rightarrow (div(a, x) \rightarrow b := \textbf{false} \mid \neg\, div(a, x) \rightarrow b := \textbf{true})|\\ &\ b = \textbf{false} \rightarrow \textbf{null}:(b; b))\end{aligned}$$

and from (32), we finally obtain the following program which has the p-type $(b;x)$:

$$\begin{aligned}&\textbf{let } y \leftarrow x \textbf{ in } \mu\xi:(b; x, y).(\\ &\ y = 2 \rightarrow b := \textbf{true}|\\ &\ 2 < y \rightarrow \text{let } y \leftarrow y - 1 \textbf{ in } \xi;\\ &\quad (b = \textbf{true} \rightarrow (div(y, x) \rightarrow b := \textbf{false} \mid \neg\, div(y, x) \rightarrow b := \textbf{true}) \mid\\ &\quad\ b = \textbf{false} \rightarrow \textbf{null}:(b; b))).\end{aligned}$$

If *div* is not executable, then we continue convert the proof Π in order to replace $div(a, x) \vee \neg\, div(a, x)$ by another $\tilde{\exists}$ formula.

§ 7. Concluding Remarks

In the present section, we shall give several remarks on our approach.

Our approach is strongly motivated by the following points:

(A) If an extracted program can be reduced to a simpler program, there should be a proof corresponding to the latter.

So we carefully pay attention to names of variables (in fact, variable names used in a proof and those in an extracted program do in most cases coincide). We also adopt transfinite induction as an induction scheme, because we think it reflects the structure of recursion. As explained in 6.4, we consider that usual mathematical induction and structural induction are not suitable for representing loop or recursion structures.

(B) We can specify variables which we do not want to appear in extracted programs.

We have two kinds of variables, marked and unmarked ones. If some variables are not desirable to appear in the extracted program, it is sufficient to use unmarked variables for them.

Due to (A) and (B), it may be possible to transform

a proof which is easily understandable but produces an inefficient program

into

a proof which is hard to understand but uses less variables and produces an efficient program.

So it is important to study such a transformation technique.

(C) Programming (or problem solving) in abstract domains is important.

Programming in an abstract domain and then realizing it in another (low level) abstract domain or some concrete domain is an important methodology to develop reliable programs. Specifications and proofs in abstract theories correspond to programming in abstract domain. In our approach, requirements for base theories are almost nothing except that the induction principle to affect program execution must be the transfinite induction. So, for example, we can use a set theory as an underlying logic providing that some executable set operation are specified.

Next we state some possibilities of extensions of our approach.

In Program Extraction Theorem, the condition (ii) is not so restrictive. From normalization theorem, the following proposition holds.

Proposition 7.1. *For every proof in* $\tilde{\exists}$**-UL0**, *if all the formulas in the assumptions, induction formulas and instances of axioms are* $\tilde{\exists}$*-SP, then we can find a proof which has the same assumptions and conclusion but all the formulas occurring in it are* $\tilde{\exists}$*-SP.*

However it is convenient if we can use formulas of the form $\tilde{\exists}\bar{x}.\varphi \supset \tilde{\exists}\bar{y}.\psi$ as lemmas. So we extend Program Extraction Theorem to SSNP (or SNP) proofs by allowing SSNP (or SNP) formulas as conclusions. Since formulas in

assumptions remain to be SSP (or SP), in the rules which have the same formula as an assumption and a premise, such as **let**-E and Ind($\prec$, $\lambda\bar{x}.\chi$), only the SSP (or SP) formula can occur as the major premise (in **let**-E) or the conclusion (in Ind). As a result, we can not use $\forall$-E for SNP formulas essentially. However $\forall$-EV can be used. By the similar reason considered for Theorem 6.1, we add $\forall$-E_0 and $\tilde{\exists}$-I_0 with some restrictions to $\tilde{\exists}$-UL1 in SNP cases. Restrictions are roughly as follows: if a term is substituted for a variable occurring in $\tilde{\exists}$-formula in a negative position, then the term must be a variable.

As a program language, PLi only has fundamental constructs, which is necessary to extract programs from proofs. In order to use PLi to analyze various kinds of programs, we need to extend PLi for *function* and *procedure* constructs.

For example, a Pascal like procedure *fact* can be defined as follows:

$$fact(x,\ \mathbf{var}\ z) \triangleq \mu\xi{:}(z;x).(x = 0 \to z := 1 \mid (\mathbf{let}\ x \leftarrow x - 1\ \mathbf{in}\ \xi{:}(z;x)); z := x \cdot z).$$

Here **let** $x \leftarrow x - 1$ **in** $\xi{:}(z; x)$ corresponds to a procedure call $fact(x - 1, z)$ in Pascal.

UL0 is an ordinary first-order predicate logic and our approach does not much depend on a specific logic and can be applied to most of usual logic. Moreover we can easily extend our approach to handle programs which accept functions as arguments. Note that this does not mean programs can be passed through parameters. We now discuss several possibilities of extensions.

In actual proofs of specifications, we need to prove various relations satisfy the well-foundedness. That is, we need to prove TI($\prec$, $\lambda\bar{x}.\chi$), when we use Ind($\prec$, $\lambda\bar{x}.\chi$) rule unless it is an axiom. To do this, we extend the underlying logic UL to a second-order logic UL^2. In UL^2, TI($\prec$, $\lambda\bar{x}.\chi$) can be expressed as follows:

$$TI(\prec, \lambda\bar{x}.\chi) \triangleq \forall p.(\forall\bar{x}.(\chi \wedge \forall\bar{y}.(\chi^{\bar{y}}_{\bar{x}} \wedge \bar{y} \prec \bar{x} \supset p(\bar{y})) \supset p(\bar{x})) \supset \forall\bar{x}.(\chi \supset p(\bar{x})))$$

For example, transfinite induction for natural numbers is proved in the second-order Peano arithmetic.

We may also want to use a ω-logic to formalize data structures completely. If we restrict ω rules only for $\tilde{\exists}$-free formulas, our approach can be straightforwardly applied to UL^{ω}, UL with ω rules, too.

Acknowledgment

The author would like to thank Professor Satoru Takasu at Research Institute of Mathematical Sciences at Kyoto University and Professor Masahiko Sato at Tohoku University for helpful discussion and encouraging him with a lot of patience. He is also grateful to Professor Nobuo Yoneda for a critical

reading of the manuscript.

References

[1] Beeson, M. J., Proving programs and programming proofs, in: R. Barcan Marcus, G. J. W. Dorn and P. Weingartner, eds., *Logic, Methodology, and Philosophy of Sciences* VII, North-Holland, Amsterdam, (1985) 51–82.

[2] ———, *Foundations of constructive mathematics*, Springer-Verlag, Berlin/Heidelberg/New York/Tokyo 1985.

[3] Constable, R. L. *et. al.*, *Implementing mathematics with the Nuprl proof development system*, Prentice-Hall, New Jersey 1986.

[4] Feferman, S., Constructive theories of functions and classes, *Logic colloquium '78*, North-Holland, Amsterdam, 159–224.

[5] Gentzen, G., Investigations into logical deduction, in: M. E. Szabo ed., *The collected works of Gehard Gentzen*, (North-Holland, Amsterdam (1969), 529–581.

[6] Gödel., K., Uber eine bisher noch nicht Erweiterung des Finten Standpunkter, *Dialectica*, **12** (1958), 208–287.

[7] Goldblatt, R., *Axiomatizing the logic of computer programming*, Lectrure Notes in Comput. Sci., **130**, Springer-Verlag, Berlin/Heidelberg/New York, 1981.

[8] Nordström, B. and Petersson, K., Programming in constructive set theory: some examples, *Proceedings of* 1981 *Conference on Functional Programming Language and Computer Architecture*, 141–153.

[9] Hayashi, S., **PX**: a system extracting programs from proofs, *Proceedings of* 3*rd Working Conference on the Formal Description of Programming Concepts*, Ebburup, Denmark, *to appear*, North-Holland, Amsterdam.

[10] Hoare, C. A. R., An axiomatic basis for computer programming, *Comm. ACM*, **12** (1969), 576–580.

[11] Igarashi, S., A natural deduction system for assertions, *preprint*, (1974).

[12] Lifschitz, V., Calculable Natural Numbers, in: S. Shapiro ed., *Intentional mathematics*, North-Holland, Amsterdam 1985,

[13] Manna, Z. and Waldinger, R., A deductive approach to program synthesis, *ACM Trans. Program. Lang. Syst.*, **2** (1980), 90–121.

[14] Martin-Löf, P., in: L. J. Cohen et al., eds., Constructive mathematics and computer programming, *Logic, Methodology, and Philosophy of Science* VI, North-Holland, Amsterdam (1982), 153–179.

[15] Prawitz, D., *Natural deduction*, Almqvist & Wiksell, Stockholm, 1965.

[16] Pratt, V.R., Semantical considerations on Floyd-Hoare logic, *Proc.* 17*th Annual IEEE Symposium on Foundations of Computer Science*, (1976) 109–121.

[17] Sato, M., Towards a mathematical theory of program synthesis, *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, (1979), 757–762.

[18] Takasu, S., Proof and Programs, *Proceedings of the* 3*rd IBM Symposium on Mathematical Foundations of Computer Science*, IBM Japan, 1978.

[19] Takasu, S. and Kawabata, S., A logical basis for programming methodology, *Theoret. Comput. Sci.*, **49** (1981), 730–753.

[20] Takasu, S., and Nakahara, T., Programming with mathematical thinking, in: *IFIP* 83, North-Holland, Amsterdam (1983), 419–424.

[21] Tatsuta, M., *Program synthesis using realizability*, Master Thesis, University of Tokyo, 1987.