Hoare Logics for Programming Languages with Partial Functions and Non-deterministic Choice

By
Likang Zhu, B.Eng.

A Thesis
Submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of

Master of Science
Department of Computing and Software
McMaster University

MASTER OF SCIENCE (2003)
(Computing and Software)

McMaster University
Hamilton, Ontario

TITLE:
Hoare Logics
for Programming Languages with Partial Functions and Non-deterministic Choice

AUTHOR: Likang Zhu, B.Eng.(Central-South University, China)

SUPERVISOR: Dr. Jeffery Zucker

NUMBER OF PAGES: vii, 182

Abstract

We develop Hoare logics for total correctness for a programming language, extending the while language, over abstract many-sorted algebras, with the following features:

(1) The algebras are partial, *i.e.*, terms are not always defined.

(2) The language includes a non-deterministic choice construct choose $\mathsf{z} : b$, where z has type nat, and b is a Boolean term, and execution diverges if there is no such z.

This language is important in the study of computation on topological partial algebras.

We develop two different logics for the assertion language, to deal with undefined truth values:

(1) the assertions extend the Booleans, and are 3-valued (including the undefined value);

(2) the assertions are disjoint from the Booleans, and are 2-valued (without undefinedness).

These lead to two distinct (but similar) Hoare logics.

Our Hoare proof rule for the choose construct seems to be original:

$$\{\exists \mathsf{z} : b\}\ \mathsf{choose}\ \mathsf{z} : b\ \{b\}$$

We prove soundness for the Hoare system in both logics, and apply it to a case study: solving a linear system of equations by Gaussian elimination.

Acknowledgements

I would like to thank Dr. J.I. Zucker, my supervisor, for his many suggestions and constant support during this research, and for his invaluable guidance at all stages of my studies.

I want to thank Dr. J. Carette and Dr. A. Wassyng for their careful review and valuable comments on my thesis. Thanks to Dr. W. M. Farmer, Dr. S. Qiao, Dr. E. Sekerinski and all my other professors for their help in my studies.

I am grateful to my parents for their patience and love. Without them this work would never have come into existence. Special thanks to Yanling Ma and Qinglan Li for their help, understanding and support.

Hamilton, Ontario Likang Zhu

September, 2003

Contents

Chapter 1

Introduction

This chapter presents a brief introduction to the background, motivation and outline of our research.

1.1 Hoare logic

The mathematical theory of program verification is a central concern in mathematics and computer science. Its origins can be traced back to Turing [Tur49]. Floyd [Flo67] assigned meanings to programs and proved correctness of flowchart programs by means of assertions. Inspired by Floyd, Hoare presented the necessary axioms and inference rules for reasoning about simple deterministic programs. Hoare's landmark paper [Hoa69] served as a pedagogical example of approaching verification of programs by *axiomatic reasoning.*

With Hoare's approach, we must first formally define the underlying program language. Next, we need a formalism that makes it possible to express the relevant program properties. In the context of program verification, the program properties are expressed by expressions in first-order logic called *assertions.* In contrast to program booleans, assertions include quantifiers, and hence are not computable in general. Finally, we need a *proof system* consisting of *axioms* and *inference rules* which allow us to construct formal proofs of certain relevant *formulas.*

In Hoare logic, these formulas are *Hoare triples*, denoted by $\{p\}S\{q\}$, where S is a program statement and p, q are assertions. Hoare triples are used to specify the input/output relation of the statements. They form the basic unit in program verification.

The triple $\{p\}S\{q\}$ is interpreted in the sense of *partial correctness* as: *if* p is true before the initiation of S and the execution of S terminates, *then* q will be true after execution of S. It is interpreted in the sense of *total correctness* as: *if* p is true before the initiation of S, *then* the execution of S terminates and q will be true after execution of S.

In this thesis, we work with *total* correctness rather than *partial* correctness, even though this is more difficult mathematically, since it seems to be a more valuable concept for the purpose of program design, specification and verification.

1.2 Topological partial algebras; Continuity

The algebra of data serves as an initial step in the formalization of a program language. A *signature* Σ (for a many-sorted partial algebra) is a pair consisting of (1) a finite set $\boldsymbol{Sort}(\Sigma)$ of *sorts* s, and (2) a finite set $\boldsymbol{Func}\,(\Sigma)$ of *typed function symbols* F. A partial many-sorted algebra A of signature Σ consists of : (1) for each sort s of Σ, a non-empty *carriers set* A_s of sort s, and (2) for each function symbol F of Σ of type $s_1 \times \cdots \times s_m \to s$, a finite family of *partial functions* of the form

$$\mathsf{F}^A : A_{s_1} \times \cdots \times A_{s_m} \overset{\cdot}{\to} A_s.$$

We will study computations on *topological algebras*, *i.e.*, many-sorted algebras in which each carrier is a topological space, such that the primitive functions are all continuous. Important examples of such algebras are algebras over reals, which will be used in our case study. A basic principle for topological algebras is the *Continuity Principle* [TZ04]:

$$\textit{Computability} \quad \Rightarrow \quad \textit{Continuity}.$$

1.3 Motivation and Objective

The original Hoare Logic [Hoa69] was designed for a simple sequential *deterministic* program language with assignment, conditionals, and 'while' statement. Classical 2-valued logic was used in program booleans, assertions and Hoare formulas because of a *totality assumption,i.e.*, all the functions of the algebra A were assumed to be *total* functions.

However, in working with topological algebras, we have to consider:

(1) *Partial functions.* To illustrate the problem, consider the assertion:

$$(x \neq 0) \;\wedge\; (y = 1/x) \quad \vee \quad (x = 0) \;\wedge\; (y = z). \tag{1.1}$$

The usual rules for evaluating (1.1) requires evaluation of $1/x$ and $y = z$ first. However, the function

$$\mathsf{div}_R : \mathbb{R}^2 \to \mathbb{R}$$

defined by

$$\mathsf{div}_R(x, y) \quad = \quad x/y$$

is *essentially partial*, since there are no *total continuous* extension of div_R, and hence (by the Continuous Principle), there is no *total computable* functions on $\mathbb{R}^2$ which extends div_R. (See [TZ04] for a detailed discussion of these issues.) Note, by contrast, the function

$$\mathsf{div}_N : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$$

defined by

$$\mathsf{div}_N(x, n) \quad = \quad x/n$$

can easily be extended to a total continuous (and computable) function by defining (say)

$$\mathsf{div}_N(x, 0) \quad = \quad 0.$$

Secondly, the function

$$\mathsf{eq}_R : \mathbb{R}^2 \to \mathbb{B}$$

and

$$\mathsf{less}_R : \mathbb{R}^2 \to \mathbb{B}$$

cannot be total and computable, since all total continuous boolean-valued functions on the reals must be constant. Hence we must define it as a partial function.

(2) *Multivalued functions.* We must consider computable functions that are both continuous and multivalued. In particular, multivalued functions are needed even to compute single valued functions. We consider an example taken from [TZ04] . Define the *partial* function

$$\text{piv} \ : \mathbb{R}^n \ \overset{\cdot}{\longrightarrow} \ \{1, \ldots, n\}$$

by

$$\text{piv}(x_1, \ldots, x_n) \ \simeq \ \begin{cases} \text{some } i : \ x_i \neq 0 & \text{if such an } i \text{ exists} \\ \uparrow & \text{otherwise.} \end{cases}$$

It can be shown that there is no *single-valued* function which satisfies the definition of piv and is *continuous* on $\mathbb{R}^n$. For such a function, being continuous and integer-valued, would have to be constant on its domain $\mathbb{R}^n \backslash \{0\}$, with constant value (say) $j \in \{1, \ldots, n\}$. But its value on the x_j-axis would have to be different from j, leading to a contradiction (This example forms the basis for our case study in Chapter 9).

Hence, in sum, the purpose of our study is to provide a logical basis for the proofs of the properties of *nondeterministic* programs with *undefinedness* on *many sorted* algebra A.

1.4 Discussion: Two types of partiality

We must distinguish between two notions of partial function:

(1) Those that can be extended to total functions.

In this case, the domain of definition is decidable, and these functions can be extended to total functions by giving a *default value*, *e.g.*, factorial function on $\mathbb{Z}$, the function $\mathsf{div}_{\mathbb{N}}$ considered above, and the array application $\mathsf{Ap}(\mathsf{a}, \mathtt{i})$ (§2.6) etc. We consider these functions as *total* functions with *error cases*.

(2) Those that cannot be extended to total functions

This case is related to topological considerations. In this case, the domain of definition is *not* computable, and these functions can not be extended to *total*, *continuous* functions, *e.g.*, the function $\mathsf{div}_R, \mathsf{eq}_R$ and less_R considered above. The issue here is *divergence* rather than halting in an *error* state. As a running example, we use the topological algebra $\mathcal{R}_p^N$ of the reals with partial operations: equality, order and diversion.

1.5 Three-tiered logic

Our Hoare style proof system contains three tiers or levels of logic:

(1) program booleans (or tests),

(2) assertions,

(3) correctness (Hoare) formulae.

The logic in boolean (tier 1) is partial and 3-valued. The logic in formulae is total and 2-valued. The middle tier (the logic in assertions) is either 3-valued, extending the boolean logic (in ***PPL***), or 2-valued conforming to the logic of formulae (in ***TPL***). The distinction between the assertion logic and Hoare formulae logic in ***PPL*** and combine them into a whole system seems to be original.

1.6 Historical remarks

(1) *Hoare style logic*

Cook [Coo78] gave the first mathematical analysis of a logical system for program verification. De Bakker [dB80] gives a detailed and rigorous treatment of verification for the basic programming language constructs. J.V. Tucker and J.I. Zucker [TZ88] developed Hoare Logic for many-sorted algebras with error cases. A concise introduction and survey of the theory of the correctness for deterministic programs is [Apt81]. This work is extended to cover nondeterministic and concurrent constructs in [Apt84, AO91].

(2) *Logic for partial functions*

Many logics for partial functions (usually in the sense of "error case", *cf.* §1.4) have been proposed, and the issue of such logics is not settled, with many problems of philosophy and style remaining. In general, there are two different approaches. One approach retains 2-valued logic. This approach has been explored in [Far90, Owe93, Par93, Gri97] etc. Another approach extends the conventional 2-valued logic to 3-valued logic. This approach has been explored in [BCJ84, Hoo87, Jon86, Jon87, KK94], continuing the work of [Luk70, Kle52, McC63].

(3) *Nondeterministic programs*

Many nondeterministic programming model have been developed. For example, Dijkstra [Dij76] proposed a *Guarded Command Language.* Another nondeterministic programming language is the *While language with random assignments* [AO91]. A *While programming language with "countable choices"* over many-sorted algebras was studied in [TZ04]. This is the language we will use for our investigation of Hoare logic.

1.7 New features in the thesis

We investigated two approaches to the logic of assertions and booleans with partial functions (*cf.* §1.5), and correspondingly proposed two Hoare style logics, which simultaneously deal with *partiality* of functions and *non-determinism.* These are *total* (2-valued) and *partial* (3-valued) predicate logics. In *total predicate logic* (***TPL***), the booleans are disjoint from the assertions, and in *partial predicate logic* (***PPL***), the booleans form a subset of the assertions. We work through the case of ***TPL*** in detail and (for the most part) indicate the requisite modifications for ***PPL***. The two resulting 3-tiered logics (§1.5) seem to be original.

Our version of *non-determinism* uses the 'choose' construct

$$\mathsf{choose}\ \mathsf{z} : b$$

which will lead to divergence in the case that there is no k such that $b(k)$ is true. Our proof rules for 'choose'

$$\{\ \exists \mathsf{z}\ \ b\ \}\quad \mathsf{choose}\ \mathsf{z} : b\quad \{\ b\ \}$$

(in ***PPL***) or

$$\{\ \exists \mathsf{z}\ \ (b\ =\ \mathsf{true})\ \}\quad \mathsf{choose}\ \mathsf{z} : b\quad \{\ (b\ = \mathsf{true})\ \}$$

(in ***TPL***) also seems to be original.

1.8 Overview of the chapters

This thesis has ten chapters.

Chapter 1 forms the introduction.

Chapter 2 presents the basic function and algebraic notions required for this thesis. We first defined standard signatures and (partial) algebras, then show how to expanded these in turn to *N-standard* signatures and algebras, and then to *array* signatures and algebras. We concentrate on topological partial algebras of the real numbers, which is used in most of our examples.

Chapter 3 introduces the syntax and semantics of a simple *deterministic* programming language $\boldsymbol{SL}(\Sigma)$ ("straight line") over a partial algebra A. We introduce partial term and statement semantics by using of *Kleene equality*.

Chapter 4 defines the *partial predicate logic* $\boldsymbol{PPL}(\Sigma)$. We give a Hoare style logic $\boldsymbol{PPL}/\boldsymbol{SL}(A)$ over this and proved its soundness in the sense of total correctness.

Chapter 5 defines the *total* predicate logic $\boldsymbol{TPL}(\Sigma)$. We give another Hoare style logic $\boldsymbol{TPL}/\boldsymbol{SL}(A)$ over this and prove its soundness.

Chapter 6 introduces the syntax and semantics of a *nondeterministic* programming language $\boldsymbol{SLCC}(\Sigma)$ ("straight line with countable choice"). We give the proof system $\boldsymbol{TPL}(\Sigma)/\boldsymbol{SLCC}$ and prove its soundness.

Chapter 7 expands $\boldsymbol{SLCC}(\Sigma)$ by adding the 'while' construct to form $\boldsymbol{WhileCC}(\Sigma)$. We give an algebraic operational semantics for nondeterministic programs with undefinedness [TZ04], and extend $\boldsymbol{TPL}(\Sigma)/\boldsymbol{SLCC}$ to $\boldsymbol{TPL}(\Sigma)/\boldsymbol{WhileCC}$ by adding a proof rule for the 'while' construct.

Chapter 8 introduced arrays. There are not really new proof rules here, since the Hoare rules for array assignments can be derived as special cases of the simple variable assignment in the array algebra A^*.

Chapter 9 presents, as a case study, a correctness proof of a $\boldsymbol{WhileCC}(\Sigma)$ program for the classical "Gaussian Elimination" problem.

Chapter 10 summarizes the result of our study and suggests some open questions for future work.

Chapter 2

Preliminaries

We give definitions and notations for partial functions and many-sorted partial algebras. This chapter is closely related to the relevant sections in [TZ00, TZ04].

2.1 Functions

Given two sets A and B, a *partial* function $f : A \overset{\cdot}{\to} B$ is a subset of $A \times B$ such that for all $a \in A$, there is *at most one* $b \in B$ (denoted $f(a)$) such that $(a, b) \in f$. We define

$$\begin{aligned}\boldsymbol{dom}(f) &= \{a \in A \mid \exists b \in B : (a, b) \in f\}\\ \boldsymbol{ran}(f) &= \{b \in B \mid \exists a \in A : (a, b) \in f\}\end{aligned}$$

A function $f : A \to B$ is *total* if $\boldsymbol{dom}(f) = A$.

In this paper, functions generally refer to *partial functions*. Totality of functions should *not* be assumed unless explicitly stated. We write partial functions as $f, g, h, \ldots$.

Notation 2.1.1. If $f : A \xrightarrow{\cdot} B$ and $x \in A$, then

$f(x)\!\uparrow$ ("$f(x)$ diverges ") means that $x \notin \boldsymbol{dom}(f)$;

$f(x)\!\downarrow$ ("$f(x)$ converges ") means that $x \in \boldsymbol{dom}(f)$;

$f(x) \downarrow y$ ("$f(x)$ converges to y") means that $x \in \boldsymbol{dom}(f)$ and $f(x) = y$, *i.e.*, $(x, y) \in f$.

Convention 2.1.2. For $f : A \to B_1 \times \ldots \times B_n$ and $f_i : A \to B_i$, we write:

$$f(x) \simeq (f_1(x), \ldots, f_n(x))$$

to mean that $f(x)\downarrow$ if and only if $f_i(x)\downarrow$ for all $i = 1, \ldots, n$, in which case $f(x) = (f_1(x), \ldots, f_n(x))$.

2.2 Many-sorted signatures

Definition 2.2.1 (Many-sorted signatures). A *many-sorted signature* Σ is a pair consisting of

(1) a finite set $\boldsymbol{Sort}(\Sigma)$ of *sorts*, and

(2) a finite set $\boldsymbol{Func}\,(\Sigma)$ of *(primitive or basic)* function symbols, each symbol F having a *type* $s_1 \times \cdots \times s_m \to s$, where $m \geq 0$ is the *arity* of F, and $s_1, \ldots, s_m \in \boldsymbol{Sort}(\Sigma)$ are the *domain sorts* and $s \in \boldsymbol{Sort}(\Sigma)$ is the *range sort*; in such a case we write

$$F : s_1 \times \cdots \times s_m \to s.$$

The case $m = 0$ corresponds to *constant symbols*; we then write

$F : \rightarrow s$ or just $F : s$.

Note that our signatures do not explicitly include relation symbols; relations will be interpreted as boolean-valued functions.

Definition 2.2.2 (Product types over Σ). A *product type* over Σ, or *Σ-product type*, is a symbol of the form $u = s_1 \times \cdots \times s_m$ $(m \geq 0)$, where $s_1, \ldots, s_m$ are sorts of Σ. We denote product types by $u, v, \ldots$.

Definition 2.2.3 (Function types over Σ). A *function type* over Σ, or *Σ-function type*, is a symbol of the form $u \rightarrow s$ where u is a Σ-product type and s a Σ-sort.

We use $\boldsymbol{Func}\,(\Sigma)_{u \to s}$ for the set of all Σ-function symbols of type $u \rightarrow s$.

2.3 Algebras A of signature Σ

Definition 2.3.1 (Σ-algebras). A *Σ-algebra* A has, for each sort s of Σ, a non-empty set A_s, called the *carrier of sort* s, and for each Σ-function symbol $F : s_1 \times \cdots \times s_m \rightarrow s$, a partial function $F^A : A_{s_1} \times \cdots \times A_{s_m} \overset{\cdot}{\rightarrow} A_s$.

For a Σ-product type $u = s_1 \times \cdots \times s_m$, we define

$$A^u =_{df} A_{s_1} \times \cdots \times A_{s_m}$$

So each Σ-function symbol $F : u \rightarrow s$ has an interpretation $F^A : A^u \overset{\cdot}{\rightarrow} A_s$. If u is empty, *i.e.*, F is a constant symbol, then F^A is an element of A_s.

The algebra A is *total* if F^A is total for each Σ-function symbol F. Without such a totality assumption, A is called *partial.*

In this paper, we deal mainly with partial algebras. The *default assumption* is that "algebra" refer to *partial algebra.* However, for the sake of emphasis, we will often speak explicitly of partial algebras.

We will sometimes write $\Sigma(A)$ for the signature of an algebra A.

We will use the following notation for signatures Σ:

$$
\boxed{
\begin{array}{lll}
\textsf{signature} & \Sigma & \\
\textsf{sorts} & & \\
 & \vdots & \\
 & s, & (s \in \boldsymbol{Sort}(\Sigma)) \\
 & \vdots & \\
\textsf{functions} & & \\
 & \vdots & \\
 & F : s_1 \times \cdots \times s_m \to s, & (F \in \boldsymbol{Func}\,(\Sigma)) \\
 & \vdots & \\
\textsf{end} & &
\end{array}
}
$$

and for Σ-structures A :

$$\boxed{\begin{array}{lll}
\textsf{algebra} & A & \\
\textsf{carriers} & & \\
 & \vdots & \\
 & A_s, & (s \in \boldsymbol{Sort}(\Sigma)) \\
 & \vdots & \\
\textsf{functions} & & \\
 & \vdots & \\
 & F^A : A_{s_1} \times \cdots \times A_{s_m} \to A_s, & (F \in \boldsymbol{Func}\,(\Sigma)) \\
 & \vdots & \\
\textsf{end} & &
\end{array}}$$

We give some examples of algebras.

Example 2.3.2. (a) The algebra of *booleans* has the carrier $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ of sort bool. The signature $\Sigma(\mathcal{B})$ and algebra $\mathcal{B}$ respectively can be displayed as follows:

$$\boxed{\begin{array}{ll}
\textsf{signature} & \Sigma(\mathcal{B}) \\
\textsf{sorts} & \mathsf{bool} \\
\textsf{functions} & \mathsf{true}, \mathsf{false} : \ \to \mathsf{bool}, \\
 & \mathsf{and}, \mathsf{or} : \mathsf{bool}^2 \to \mathsf{bool} \\
 & \mathsf{not} : \mathsf{bool} \to \mathsf{bool} \\
\textsf{end} &
\end{array}}
\quad \text{and} \quad
\boxed{\begin{array}{ll}
\textsf{algebra} & \mathcal{B} \\
\textsf{carriers} & \mathbb{B} \\
\textsf{functions} & \mathsf{tt}, \mathsf{ff} : \ \to \mathbb{B}, \\
 & \mathsf{and}^{\mathcal{B}}, \mathsf{or}^{\mathcal{B}} : \mathbb{B}^2 \to \mathbb{B} \\
 & \mathsf{not}^{\mathcal{B}} : \mathbb{B} \to \mathbb{B} \\
\textsf{end} &
\end{array}}$$

Note that the signature can essentially be inferred from the algebra; indeed from now on we will not define the signature where no confusion will arise. Further, for notational simplicity, we will not always distinguish between function names in the signature (true, etc.) and their intended interpretations ($\mathsf{true}^{\mathcal{B}} = \mathtt{tt}$, etc.).

(b) The algebra $\mathcal{N}_0$ of naturals has a carrier $\mathbb{N}$ of sort nat, together with the zero constant and successor function:

algebra	$\mathcal{N}_0$
carriers	$\mathbb{N}$
functions	$0 : \ \to \mathbb{N}$,
	$\mathsf{S} : \mathbb{N} \to \mathbb{N}$
end	

(c) The ring $\mathcal{R}_0$ of reals has a carrier $\mathbb{R}$ of sort real:

algebra	$\mathcal{R}_0$
carriers	$\mathbb{R}$
functions	$0, 1 : \ \to \mathbb{R}$,
	$+, \times : \mathbb{R}^2 \to \mathbb{R}$,
	$- : \mathbb{R} \to \mathbb{R}$
end	

(d) The field $\mathcal{R}_1$ of reals is formed by adding the multiplicative inverse to the ring $\mathcal{R}_0$:

```
algebra     R_1
import      R_0
functions   inv^R : ℝ ⇀ ℝ
end
```

where

$$\mathsf{inv}^{\mathcal{R}}(x) \;=\; \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

Note that all the examples except (d) are total algebras.

Definition 2.3.3 (Reducts and expansions). Let Σ and Σ' be signatures.

(a) $\Sigma \subseteq \Sigma'$ means $\boldsymbol{Sort}(\Sigma) \subseteq \boldsymbol{Sort}(\Sigma')$ and $\boldsymbol{Func}(\Sigma) \subseteq \boldsymbol{Func}(\Sigma')$.

(b) Suppose $\Sigma \subseteq \Sigma'$. Let A and A' be algebras with signatures Σ and Σ' respectively.

(i) The *Σ-reduct* $A'|_{\Sigma}$ *of* A' is the algebra of signature Σ, consisting of the carriers of A' named by the sorts of Σ and equipped with the functions of A' named by the function symbols of Σ.

(ii) A' is a *Σ'-expansion* of A if, and only if, A is the Σ-reduct of A'.

Definition 2.3.4 (Closed terms over Σ). We define the class $T(\Sigma)$ of *closed terms over* Σ, denoted $t, t', t_1, \ldots,$ and for each Σ-sort s, the class $T(\Sigma)_s$ of closed terms of sort s. These are generated inductively by the rule:

$$\text{If } F \in \boldsymbol{Func}\,(\Sigma)_{u \to s} \text{ and } t_i \in T(\Sigma)_{s_i} \text{ for } i = 1, \ldots, m$$

$$\text{where } u = s_1 \times \cdots \times s_m, \text{ then } F(t_1, \ldots, t_m) \in T(\Sigma)_s.$$

Note that the implicit base case of this inductive definition is the case that $m = 0$, which yields: for all constants $c : \to s$, $c() \in T(\Sigma)_s$. In this case we write c instead of $c()$. Hence if Σ contains no constants, $T(\Sigma)$ is empty.

Assumption 2.3.5 (Instantiation Assumption). Throughout this thesis we make the following assumption about the signatures Σ:
For every sort s of Σ, there is a closed term of that sort, called the default term $\boldsymbol{\delta}^s$ of sort s.

This guarantees the presence of *default values* $\boldsymbol{\delta}^s_A$ in a Σ-algebra A at all sorts s, and *default tuples* $\boldsymbol{\delta}^u_A$ at all product types u.

2.4 Adding booleans: Standard signatures and algebras

The algebra $\mathcal{B}$ of booleans plays an essential role in computation. This motivates the following definition.

Definition 2.4.1 (Standard signature). A signature Σ is *standard* if

(i) it is an expansion of $\Sigma(\mathcal{B})$, *i.e.* $\Sigma(\mathrm{B}) \subseteq \Sigma$.

(*ii*) the function symbol of Σ include an *equality operator*

$$\mathsf{eq}_s : s^2 \to \mathsf{bool}.$$

for certain sorts s, called *equality sorts.*

Definition 2.4.2 (Standard algebra). Given a standard signature Σ, a Σ-algebra A is *standard* if

(*i*) it is an expansion of $\mathcal{B}$, and

(*ii*) the operator eq_s is interpreted as a *partial identity* on each equality sort s, *i.e.*, for any two elements of A_s, if they are identical, then the operator at these arguments returns tt or $\uparrow$, and if they are not identical, it returns ff or $\uparrow$.

Remark 2.4.3. In computing with an abstract model on A, we assume A has some boolean-valued functions to test data, for example, eq_s and less_s at certain sort s. Three typical examples of partial identity as an interpretation of eq_s are:

(1) *Total equality*, where equality is assumed to be *decidable* at sort s; for example, when $s = \mathsf{nat}$:

$$\mathsf{eq}_s^A(x, y) = \begin{cases} \mathsf{tt} & \text{if } x = y \\ \mathsf{ff} & \text{otherwise} \end{cases}$$

(2) *Semi-equality*, where equality is *semi-decidable* at sort s; for example, the

initial term algebra of an equational theory:

$$\mathsf{eq}_s^A(x,y) \;=\; \begin{cases} \mathsf{tt} & \text{if } x = y \\ \uparrow & \text{otherwise} \end{cases}$$

(3) *Co-semi-equality*, where equality is *co-semidecidable* at sort s; for example, when $s = \mathsf{real}$:

$$\mathsf{eq}_s^A(x,y) \;=\; \begin{cases} \uparrow & \text{if } x = y \\ \mathsf{ff} & \text{otherwise} \end{cases}$$

The important cases for our works are (1) and (3), which will be used throughout the thesis, with the cases $s = \mathsf{nat}$ and $s = \mathsf{real}$ respectively, *i.e.*, in the algebras $\mathcal{R}_p$, $\mathcal{R}_p^N$, and $\mathcal{R}_p^{N^*}$ (Examples 2.4.5 (c), 2.5.3(b) and 2.6.2 below).

Remark 2.4.4. (a) In our terminology, an "equality sort" s need not have computable equality; it may have a *partial* equality which may be (only) semicomputable or co-semicomputable, *e.g.* $s = \mathsf{real}$, (*cf.* Discussion 2.4.6.)

(b) Any many-sorted signature Σ can be *standardised* to a signature $\Sigma^{\mathcal{B}}$ by adjoining the sort bool together with the standard boolean operations; and, correspondingly, any algebra A can be standardised to an algebra $A^{\mathcal{B}}$ by adjoining the algebra $\mathcal{B}$ as well as *equality* operators.

Example 2.4.5 (Standard algebras). (a) The simplest standard algebra is the algebra $\mathcal{B}$ of the booleans (Example 2.3.2(a)).

(b) A *standard total algebra of naturals* $\mathcal{N}$ is formed by standardising the algebra

$\mathcal{N}_0$ (Example 2.3.2(*b*)), with (total) equality and order operations on $\mathbb{N}$:

```
algebra     𝒩
import      𝒩₀, ℬ
functions
            eq^𝒩_nat, less^𝒩_nat : ℕ² → 𝔹
end
```

(*c*) A *standard partial algebra $\mathcal{R}_p$ on the reals* is formed by standardising the field $\mathcal{R}_1$ (Example 2.3.2(*d*)), with (partial) equality and order operations on $\mathbb{R}$:

```
algebra     ℛ_p
import      ℛ₁, ℬ
functions
            eq^ℛ_real, less^ℛ_real : ℝ² → 𝔹
end
```

where

$$\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}(x,y) = \begin{cases} \uparrow & \text{if } x = y \\ \mathsf{ff} & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathsf{less}^{\mathcal{R}}_{\mathsf{real}}(x,y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

This is motivated by the following discussion.

Discussion 2.4.6 (Partial equality and order on reals). Given two reals x,y

represented by infinite decimal expansions, if $x \neq y$, we can evaluate $\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}(x, y)$ and $\mathsf{less}^{\mathcal{R}}_{\mathsf{real}}(x, y)$ in finitely many steps. However, if $x = y$, we will not be able to compute this in finite many steps. (The point is that computations on infinite precision real numbers involve infinite data.) Hence, to study the full range of real number computations, we must define $\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}(x, y)$ and $\mathsf{less}^{\mathcal{R}}_{\mathsf{real}}(x, y)$ as *partial* boolean-valued functions.

By the continuity principle:

$$Computable \quad \Rightarrow \quad Continuous$$

we find that total equality on $\mathbb{R}$ (case 1 above) is *not* continuous. This is because the only continuous total functions from a connected space ($\mathbb{R}^2$) to a discrete space are the constant functions, so the total continuous boolean-valued functions on the reals must be constant. Therefore, total equality on $\mathbb{R}$ is not computable. But the *partial* equality and order operations (Example (c) above) are continuous and co-semicomputable. ([TZ04] has a thorough discussion of these issues.)

2.5 Adding counters: N-standard signatures and algebras

Definition 2.5.1. (a) A standard signature Σ is called *N-standard* if it includes (as well as bool) the *numerical sort* nat, and also function symbols for the *standard operations* of *zero*, *successor* and *order* on the naturals:

$0 : \ \rightarrow \mathsf{nat}$

$\mathsf{S} : \mathsf{nat} \to \mathsf{nat}$

$\mathsf{less}_{\mathsf{nat}} : \mathsf{nat}^2 \to \mathsf{bool}$

as well as the *equality operator* $\mathsf{eq}_{\mathsf{nat}}$ on nat.

(b) The corresponding Σ-algebra A is *N-standard* if the carrier A_{nat} is the set of natural numbers $\mathbb{N}= \{0,1,2,\ldots\}$, and the standard operations (listed above) have their *standard interpretations* on $\mathbb{N}$.

Definition 2.5.2. (a) The N-*standardisation* Σ^N of a standard signature Σ is formed by adjoining the sort nat and the operations 0, S, $\mathsf{eq}_{\mathsf{nat}}$ and $\mathsf{less}_{\mathsf{nat}}$.

(b) The N-*standardisation* A^N of a standard Σ-algebra A is the Σ^N-algebra formed by adjoining the carrier $\mathbb{N}$ together with its standard operations to A, thus:

algebra	A^N
import	A
carriers	$\mathbb{N}$
functions	$0 : \ \to \mathbb{N}$
	$\mathsf{S} : \mathbb{N} \to \mathbb{N}$
	$\mathsf{eq}_{\mathsf{nat}}, \mathsf{less}_{\mathsf{nat}} : \mathbb{N}^2 \to \mathbb{B}$
end	

Example 2.5.3. (a) The simplest N-standard algebra is the algebra $\mathcal{N}$ of Example 2.3.2(b).

(b) We can N-standardise $\mathcal{R}_p$ (Example 2.4.5(c)) by adjoining the carrier $\mathbb{N}$ together with standard operations on $\mathcal{R}_p$:

algebra	$\mathcal{R}_p^N$
import	$\mathcal{R}_p$
carriers	$\mathbb{N}$
functions	$0 : \ \to \mathbb{N}$
	$\mathsf{S} : \mathbb{N} \to \mathbb{N}$
	$\mathsf{eq}_{\mathsf{nat}}, \mathsf{less}_{\mathsf{nat}} : \mathbb{N}^2 \to \mathbb{B}$
end	

Assumption 2.5.4 (N-standardness Assumption). Throughout this thesis, we will assume:

The signatures Σ, and the Σ-algebra A, are N-standard.

2.6 Adding arrays: Algebras A^* of signature Σ^*

Given an N-standard signature Σ, and N-standard Σ-algebra A, we extend Σ, and expand A, as follows:

Define, for each sort s of Σ, the carrier A_s^* to be the set of *finite sequences* or *arrays* a^* over A_s, of "starred sort" s^*.

The resulting algebras A^* have signature Σ^*, which extends Σ^N by including, for each sort s of Σ, the new starred sorts s^*, and also the following new function symbols:

(i) the operator $\mathsf{Lgth}_s : s^* \to \mathsf{nat}$, where $\mathsf{Lgth}(a^*)$ is the length of the array a^*;

(ii) the application operator $\mathsf{Ap}_s : s^* \times \mathsf{nat} \to s$, where

$$\mathsf{Ap}_s^A(a^*, k) \;=\; \begin{cases} a^*[k] & \text{if } k < \mathsf{Lgth}(a^*) \\ \boldsymbol{\delta}^s & \text{otherwise} \end{cases}$$

where $\boldsymbol{\delta}^s$ is the default value at sort s (§2.1);

(iii) the null array $\mathsf{Null}_s : s^*$ of zero length;

(iv) the operator $\mathsf{Update}_s : s^* \times \mathsf{nat} \times s \to s^*$, where $\mathsf{Update}_s^A(a^*, n, x)$ is the array $b^* \in A_s^*$ of length $\mathsf{Lgth}(a^*)$ such that for all $k < \mathsf{Lgth}(a^*)$,

$$b^*[k] = \begin{cases} a^*[k] & \text{if } k \neq n \\ x & \text{if } k = n \end{cases}$$

(v) the operator $\mathsf{Newlength}_s : s^* \times \mathsf{nat} \to s^*$, where $\mathsf{Newlength}_s^A(a^*, m)$ is the array b^* of length m such that for all $k < m$,

$$b^*[k] = \begin{cases} a^*[k] & \text{if } k < \mathsf{Lgth}(a^*) \\ \boldsymbol{\delta}^s & \text{otherwise.} \end{cases}$$

(vi) the *equality* operator on A_s^* for each equality sort s.

Remark 2.6.1. (a) Note that A^* is an N-standard *Σ^*-expansion* of A.

(b) The justification for (vi) is that if a sort s has computable or semicomputable equality, then clearly so has the sort s^*, since it amounts to testing equality of finitely many pairs of objects of sort s, up to a computable length.

(c) The reason for introducing starred sorts is the lack of effective coding of finite sequences within abstract algebras in general.

(d) The significance of arrays for computation is that they provide *finite but unbounded memory.*

(e) We prefer to make Ap a *total* rather than a *partial* function using the default values (see the discussion in §1.4).

Example 2.6.2 (N-standard partial algebra of reals with array). Recall the algebra $\mathcal{R}_p^N$ (Example 2.5.3(b)). We construct $\mathcal{R}_p^{N^*}$ as follows:

algebra	$\mathcal{R}_p^{N^*}$
import	$\mathcal{R}_p^N$
carriers	$\mathcal{R}^*$
functions	$\mathsf{Null}_{\mathsf{real}} : \ \to \mathbb{R}^*$
	$\mathsf{Ap}_{\mathsf{real}} : \mathbb{R}^* \times \mathbb{N} \to \mathbb{R}$
	$\mathsf{Update}_{\mathsf{real}} : \mathbb{R}^* \times \mathbb{N} \times \mathbb{R} \to \mathbb{R}^*$
	$\mathsf{Lgth}_{\mathsf{real}} : \mathbb{R}^* \to \mathbb{N}$
	$\mathsf{Newlength}_{\mathsf{real}} : \mathbb{R}^* \times \mathbb{N} \to \mathbb{R}^*$
	$\mathsf{eq}_{\mathsf{real}} : (\mathbb{R}^*)^2 \to \mathbb{B}$
end	

The N-standard partial algebra $\mathcal{R}_p^{N^*}$ is an important example for the theory developed in this thesis and will be used in our examples, *e.g.* in the case study in Chapter 9.

Chapter 3

Straight-Line Programs $\boldsymbol{SL}(\Sigma)$

In this chapter, we will study the syntax and semantics of the imperative ***Straight-Line*** programming language (written as $\boldsymbol{SL}(\Sigma)$) on N-standard partial Σ-algebras. This model takes into account *undefinedness* in the programming language. Unlike the $\boldsymbol{SLCC}(\Sigma)$ language (Chapter 6), $\boldsymbol{SL}(\Sigma)$ is *deterministic*, *i.e.* from a given initial state only one execution sequence is generated. We then introduce *Kleene equality* and use it to generalize the functionality and substitution lemma for program terms [SA91, TZ00] to *partial algebras*.

Assume Σ is an N-standard signature, and A is an N-standard partial Σ-algebra.

3.1 Syntax of $\boldsymbol{SL}(\Sigma)$

We define four syntactic classes: *variables*, *terms*, *statements* and *procedures*.

(a) $\boldsymbol{Var} = \boldsymbol{Var}(\Sigma)$ is the class of *Σ-program variables*, and for each Σ-sort s, $\boldsymbol{Var}_s$ is the class of program variables of sort s: $\mathtt{x}^s, \mathtt{y}^s \ldots$.

(b) $\boldsymbol{Term} = \boldsymbol{Term}(\Sigma)$ is the class of *Σ-program terms* $t, \ldots,$ and for each Σ-sort s, $\boldsymbol{Term}_s$ is the class of program terms of sort s. We sometimes write $t : s$ or t^s to indicate that t has sort s. Program terms are defined by:

$$t^s \ ::= \ \mathsf{x}^s \mid \mathsf{F}(t_1, \ldots, t_m) \mid \mathsf{if} \ \ b \ \mathsf{then} \ \ t_1^s \ \mathsf{else} \ \ t_2^s \ \mathsf{fi}$$

where $s, s_1, \ldots, s_m$ $(m \geq 0)$ are Σ-sorts, F is a Σ-function symbol of type $s_1 \times \cdots \times s_m \to s$. $t_i \in \boldsymbol{Term}_{s_i}$ for $i = 1, \ldots, m$, and b is a boolean term, *i.e.* a term of sort bool. For the sake of clarity, we repeat the definition of $\boldsymbol{Term}_s$ for $s = \mathsf{bool}$ (writing b for t^{bool}).

$$\begin{aligned} b \ ::= \ & \mathsf{true} \mid \mathsf{false} \mid \mathsf{not} \ b \mid b_1 \ \mathsf{and} \ b_2 \mid b_1 \ \mathsf{or} \ b_2 \\ & \mid \mathsf{F}(t_1, \ldots, t_m) \mid \mathsf{if} \ b \ \mathsf{then} \ b_1 \ \mathsf{else} \ b_2 \ \mathsf{fi} \end{aligned}$$

where $\mathsf{F} : s_1 \times \cdots \times s_m \to \mathsf{bool}$ is a Σ-function symbol (other than $\mathsf{not}, \mathsf{and}, \mathsf{or}$). Note that F maybe the equality function $\mathsf{eq}_s : s^2 \to \mathsf{bool}$ at equality sorts s.

We write $t : u$ to indicate that t is a *u-tuple* of program terms, *i.e.*, a tuple of program terms of sorts $s_1, \ldots, s_m$, where $u = s_1 \times \cdots \times s_m$.

(c) $\boldsymbol{AtSt} = \boldsymbol{AtSt}(\Sigma)$ is the class of *atomic statements* $S_{\mathsf{at}}, \ldots$ defined by

$$S_{\mathsf{at}} \ ::= \ \mathsf{skip} \mid \mathtt{x} := t$$

Here $\mathtt{x} := t$ is a *concurrent assignment*, *i.e.* for some *Σ-product type u*, $t : u$ and $\mathtt{x}$ is a u-tuple of *distinct* variables.

(d) $\boldsymbol{Stmt} = \boldsymbol{Stmt}(\Sigma)$ is the class of statements $S, \ldots,$ defined by:

$$S \ ::= \ S_{\mathsf{at}} \ | \ S_1; S_2 \ | \ \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$$

(e) $\boldsymbol{Proc} = \boldsymbol{Proc}(\Sigma)$ is the class of function procedures $P, Q, \ldots.$ These have the form

$$P \ \equiv \ \mathsf{proc\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

where a, b and c are lists of *input variables*, *output variables* and *auxiliary* (or *local) variables* respectively, and S is the *body* of P. Further, we stipulate:

- a, b and c each consist of distinct variables, and they are pairwise disjoint,
- all variables occurring in S must be among a, b or c,
- *input variables* a must not occur on the lhs of assignments in S.

Each variable occurs in the declaration of a procedure *binds* all free occurrences of that variable in the body.

If $\mathsf{a} : u$ and $\mathsf{b} : v$, then P is said to have *type* $u \to v$, written $P : u \to v$. Its *input type* is u, and its *output type* is v.

We write $\boldsymbol{Proc}_{u \to v} = \boldsymbol{Proc}(\Sigma)_{u \to v}$ for the class of Σ-procedures of type $u \to v$.

Notation 3.1.1. In this thesis, $\boldsymbol{Var}$ denotes the set of variables. $\boldsymbol{Var}(t)$ denotes the set of variables that occur in term t. $\boldsymbol{Var}(S)$ denotes the set of variables that occur in statement S.

3.2 Semantics of programming terms of $\boldsymbol{SL}(\Sigma)$

We are going to extend the semantics of $\boldsymbol{SL}(\Sigma)$ [TZ88, TZ00] to *partial* algebras.

Definition 3.2.1 (State). For our N-standard Σ-algebra A, a state on A is a family

$$< \sigma_s \mid s \in \boldsymbol{Sort}(\Sigma) >$$

of functions

$$\sigma_s : \boldsymbol{Var}_s \to A_s.$$

Let $\boldsymbol{State}(A)$ be the set of states on A, with typical elements $\sigma, \ldots$

Definition 3.2.2 (Variant of state). Let σ be a state over A_s, $\mathtt{x} : s$ and $a \in A_s$. We define $\sigma\{\mathtt{x}/a\}$ to be the state over A_s formed from σ by replacing its value at $\mathtt{x}$ by a. That is , for all variables $\mathtt{y}$:

$$\sigma\{\mathtt{x}/a\}(\mathtt{y}) = \begin{cases} \sigma(\mathtt{y}) & \text{if } \mathtt{y} \not\equiv \mathtt{x} \\ a & \text{otherwise} \end{cases}$$

The following lemma on variants of states [SA91] will be used later.

Lemma 3.2.3 (Variant of state).

(1) $\sigma\{\mathtt{x}/\sigma(\mathtt{x})\} = \sigma$

(2) $\sigma\{\mathtt{x}/a_1\}\{\mathtt{x}/a_2\} = \sigma\{\mathtt{x}/a_2\}$

(3) $\sigma\{\mathtt{x}/a_1\}\{\mathtt{y}/a_2\} = \sigma\{\mathtt{y}/a_2\}\{\mathtt{x}/a_1\}$

Proof. The proofs are routine. □

We now give the semantics of the syntactic class $\boldsymbol{Term}$. For $t \in \boldsymbol{Term}_s$, we define *term evaluation* as a *partial function*

$$[\![t]\!]^A : \boldsymbol{State}(A) \stackrel{\cdot}{\to} A_s,$$

where $[\![t]\!]^A\sigma$ is the value of t in A at state σ. Our approach extends the approach for total algebras [TZ88, TZ00] to partial algebras. To represent divergence of the evaluation of a program term under a state, we use the symbol '$\uparrow$'.

Before giving the semantic definitions, we introduce and define the meaning of *Kleene equality.*

Definition 3.2.4 (Kleene equality). $[\![t_1^s]\!]^A\sigma \simeq [\![t_2^s]\!]^A\sigma'$ if and only if either both $[\![t_1^s]\!]^A\sigma\uparrow$ and $[\![t_2^s]\!]^A\sigma'\uparrow$, or both $[\![t_1^s]\!]^A\sigma\downarrow$ and $[\![t_2^s]\!]^A\sigma'\downarrow$ and $[\![t_1^s]\!]^A\sigma = [\![t_2^s]\!]^A\sigma'$.

Note that the relation $\simeq$ is an equivalence relation.

Definition 3.2.5 (Semantics of program terms). The definition is by structural induction on t , simultaneously for all Σ-sorts s.

$$[\![\mathtt{x}]\!]^A\sigma = \sigma(\mathtt{x})$$

$$[\![\mathsf{F}(t_1,\dots,t_m)]\!]^A\sigma \simeq \begin{cases} F^A([\![t_1]\!]^A\sigma,\dots,[\![t_m]\!]^A\sigma) \\ \quad \text{if } [\![t_i]\!]\sigma\downarrow \text{ for } i=1,\dots,m \\ \quad \text{and } ([\![t_1]\!]^A\sigma,\dots,[\![t_m]\!]^A\sigma) \in \boldsymbol{dom}(F) \\ \uparrow \quad \text{otherwise} \end{cases}$$

$$[\![\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}]\!]^A\sigma \;\simeq\; \begin{cases} [\![t_1]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma\downarrow \mathtt{tt} \\ [\![t_2]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma\downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

Remark 3.2.6 (Non-strictness of conditional operator). An operator is said to be *strict* if the result is always undefined when any of its arguments is undefined, and is *non-strict* otherwise.

All the primitive functions in the signature define strict operators. However the conditional operator is not strict. The concept of *strictness* and *non-strictness* of operators corresponds to two evaluation techniques: *eager* and *lazy.* In *eager evaluation* all the arguments of a function are evaluated before the function is applied. In *lazy evaluation* the components of the expression are expanded in a "demand driven" way and are not evaluated more than is necessary to provide a value at the top level. Terms built up using Σ-functions (only) are evaluated eagerly, but terms containing the *conditional* 'if - then - else - fi' construct may not. For example, 'cand' and 'cor' (see below) are evaluated lazily.

The *conditional and* ('cand') and *the conditional or* ('cor') can be defined explicitly using 'if - then - else - fi' as follows:

b_1 cand b_2 $\equiv_{df}$ if b_1 then b_2 else false fi

b_1 cor b_2 $\equiv_{df}$ if b_1 then true else b_2 fi

Note again that the current approach leads to the construction of *partial single-valued term semantics*, since our primitive functions are all single-valued, but not necessarily *total.*

For the sake of clarity again, we give the semantics of program booleans separately:

$$[\![\mathsf{true}]\!]^A\sigma \;= \mathtt{tt}$$

$$[\![\mathsf{false}]\!]^A\sigma \;= \mathtt{ff}$$

$$[\![\mathsf{not}(b)]\!]^A\sigma \;\simeq \begin{cases} \mathtt{tt} & \text{if } [\![b]\!]^A\sigma \downarrow \mathtt{ff} \\ \mathtt{ff} & \text{if } [\![b]\!]^A\sigma \downarrow \mathtt{tt} \\ \uparrow & \text{otherwise} \end{cases}$$

$$[\![b_1 \mathsf{~and~} b_2]\!]^A\sigma \;\simeq \begin{cases} \mathtt{tt} & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{tt} \;\text{ and }\; [\![b_2]\!]^A\sigma \downarrow \mathtt{tt} \\ \mathtt{ff} & \text{if } [\![b_1]\!]^A\sigma \downarrow \;\text{ and }\; [\![b_2]\!]^A\sigma \downarrow \;\text{ and }\; ([\![b_1]\!]^A\sigma \downarrow \mathtt{ff} \text{ or } [\![b_2]\!]^A\sigma \downarrow \mathtt{ff}) \\ \uparrow & \text{otherwise} \end{cases}$$

$$[\![b_1 \mathsf{~or~} b_2]\!]^A\sigma \;\simeq \begin{cases} \mathtt{tt} & \text{if } [\![b_1]\!]^A\sigma \downarrow \;\text{ and }\; [\![b_2]\!]^A\sigma \downarrow \;\text{ and }\; ([\![b_1]\!]^A\sigma \downarrow \mathtt{tt} \text{ or } [\![b_1]\!]^A\sigma \downarrow \mathtt{tt}\ ) \\ \mathtt{ff} & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{ff} \;\text{ and }\; [\![b_2]\!]^A\sigma \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

$$[\![\mathsf{F}(t_1,\ldots,t_m)]\!]^A\sigma \;\simeq \begin{cases} F^A([\![t_1]\!]^A\sigma,\ldots,[\![t_m]\!]^A\sigma) \\ \qquad \text{if } [\![t_i]\!]\sigma \downarrow \text{ for } i=1,\ldots,m \\ \qquad \text{and } ([\![t_1]\!]^A\sigma,\ldots,[\![t_m]\!]^A\sigma) \in \boldsymbol{dom}(F) \\ \uparrow \quad \text{otherwise} \end{cases}$$

$$[\![\mathsf{if}\ b \mathsf{~then~} b_1 \mathsf{~else~} b_2 \mathsf{~fi}]\!]^A\sigma \;\simeq \begin{cases} [\![b_1]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma \downarrow \mathtt{tt} \\ [\![b_2]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

Defining cand and cor from the *conditional* operator as above:

$$[\![b_1 \ \mathsf{cand}\ b_2]\!]^A\sigma \ \simeq \begin{cases} [\![b_2]\!]\sigma & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{tt} \\ \mathtt{ff} & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

$$[\![b_1 \ \mathsf{cor}\ b_2]\!]^A\sigma \ \simeq \begin{cases} \mathtt{tt} & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{tt} \\ [\![b_2]\!]\sigma & \text{if } [\![b_1]\!]^A\sigma \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

Remark 3.2.7 (Non-strict predicate logic). We give truth tables to show the difference between the *strict* operators ('and', 'or') and the *non-strict* operators ('cand' , 'cor'). First 'and' and 'cand':

b_1 and b_2	tt	ff	$\uparrow$
tt	tt	tt	$\uparrow$
ff	ff	ff	$\uparrow$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

b_1 cand b_2	tt	ff	$\uparrow$
tt	tt	ff	$\uparrow$
ff	ff	ff	ff
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

Next, 'or' and 'cor' :

b_1 or b_2	tt	ff	$\uparrow$
tt	tt	tt	$\uparrow$
ff	tt	ff	$\uparrow$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

b_1 cor b_2	tt	ff	$\uparrow$
tt	tt	tt	tt
ff	tt	ff	$\uparrow$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

Note the semantic definition of the boolean operators can be rewritten as:

$$\begin{aligned}
[\![\mathsf{not}\ b]\!]^A\sigma &= \text{not }([\![b]\!]^A\sigma)\\
[\![b_1\ \mathsf{and}\ b_2]\!]^A\sigma &= [\![b_1]\!]^A\sigma \text{ and } [\![b_2]\!]^A\sigma\\
[\![b_1\ \mathsf{or}\ b_2]\!]^A\sigma &= [\![b_1]\!]^A\sigma \text{ or } [\![b_2]\!]^A\sigma\\
[\![b_1\ \mathsf{cand}\ b_2]\!]^A\sigma &= [\![b_1]\!]^A\sigma \text{ cand } [\![b_2]\!]^A\sigma\\
[\![b_1\ \mathsf{cor}\ b_2]\!]^A\sigma &= [\![b_1]\!]^A\sigma \text{ cor } [\![b_2]\!]^A\sigma
\end{aligned}$$

In the above definition, we use ‘cand’, ‘cor’, ‘and’, ‘or’, ‘not’ to stand for meta-level logical operators.

3.3 Semantics of program statements

The semantics of statements are essentially *partial state transformations*, *i.e.*

$$[\![S]\!]^A : \boldsymbol{State}(A) \stackrel{\cdot}{\rightarrow} \boldsymbol{State}(A).$$

Note that this is a *partial function,* defined by structural induction on S:

Definition 3.3.1 (Semantics of statements).

$$[\![\mathsf{skip}\ ]\!]^A\sigma = \sigma$$

$$[\![\mathtt{x} := t]\!]^A\sigma \simeq \begin{cases} \sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} & \text{if } [\![t]\!]^A\sigma\downarrow\\ \uparrow & \text{otherwise}\end{cases}$$

$$[\![S_1; S_2]\!]^A\sigma \simeq \begin{cases} [\![S_2]\!]^A([\![S_1]\!]^A\sigma) & \text{if } [\![S_1]\!]^A\sigma\downarrow\\ \uparrow & \text{otherwise}\end{cases}$$

$$[\![\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}]\!]^A\sigma \ \simeq \begin{cases} [\![S_1]\!]^A\sigma & \text{if} \ \ [\![b]\!]^A\sigma \downarrow \mathsf{tt} \\ [\![S_2]\!]^A\sigma & \text{if} \ \ [\![b]\!]^A\sigma \downarrow \mathsf{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

Definition 3.3.2 (Substitution for terms). We define $t'[x/t]$ by structural induction on t' [SA91].

$$\mathtt{x}[\mathtt{y}/t] \ \equiv \begin{cases} t & \text{if} \quad \mathtt{x} \equiv \ \mathtt{y} \\ \mathtt{x} & \text{otherwise} \end{cases}$$

$$\mathsf{F}(t_1, \ldots, t_m)[\mathtt{x}/t] \ \equiv \ \mathsf{F}(t_1[\mathtt{x}/t], \ldots, t_m[\mathtt{x}/t])$$

$$(\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}\ )[\mathtt{x}/t] \ \equiv \ \mathsf{if}\ b[\mathtt{x}/t]\ \mathsf{then}\ t_1[\mathtt{x}/t]\ \mathsf{else}\ t_2[\mathtt{x}/t]\ \mathsf{fi}$$

Definition 3.3.3 (Substitution for statements).

$$(\mathtt{x} := t)[\mathtt{y}/\mathtt{z}] \ \equiv \ \mathtt{x}[\mathtt{y}/\mathtt{z}] := t[\mathtt{y}/\mathtt{z}]$$

$$(S_1; S_2)[\mathtt{y}/\mathtt{z}] \equiv S_1[\mathtt{y}/\mathtt{z}]; S_2[\mathtt{y}/\mathtt{z}]$$

$$(\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi})[\mathtt{y}/\mathtt{z}] \ \equiv \ \mathsf{if}\ b[\mathtt{y}/\mathtt{z}]\ \mathsf{then}\ S_1[\mathtt{y}/\mathtt{z}]\ \mathsf{else}\ S_2[\mathtt{y}/\mathtt{z}]\ \mathsf{fi}$$

Definition 3.3.4 (Equivalence of states relative to a set of variables). For $M \subseteq \boldsymbol{Var}$, $\sigma \approx \sigma'(\mathsf{rel}\ M)$ iff $\sigma \restriction M \ = \ \sigma' \restriction M$, *i.e.*,

$$\text{for all } \mathtt{x} \in M, \quad \sigma(\mathtt{x}) = \sigma'(\mathtt{x})$$

Lemma 3.3.5 (Functionality lemma for terms).

If $\sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(t))$ then $[\![t]\!]^A\sigma \simeq [\![t]\!]^A\sigma'$

Proof. Suppose $\sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(t))$. We want to show

$[\![t]\!]^A\sigma \simeq [\![t]\!]^A\sigma'$.

The proof is by structural induction on t.

- $t \equiv \mathtt{x}$

  Since $\sigma \restriction \boldsymbol{Var}(\mathtt{x}) = \sigma' \restriction \boldsymbol{Var}(\mathtt{x})$,

  so $[\![\mathtt{x}]\!]^A\sigma = \sigma(\mathtt{x}) = \sigma'(\mathtt{x}) = [\![\mathtt{x}]\!]^A\sigma'(\mathtt{x})$.

- $t \equiv \mathsf{F}(t_1 \ldots t_m)$

  By i.h. (induction hypothesis),

  Since $\sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(t_i))$,

  therefore $[\![t_i]\!]^A\sigma \simeq [\![t_i]\!]^A\sigma'$, for $i = 1, \ldots, m$.

  Suppose for all $i \in \{1, \ldots, m\}$,

  $[\![t_i]\!]^A\sigma \downarrow$ and $([\![t_1]\!]^A\sigma, \ldots, [\![t_m]\!]^A\sigma) \in \boldsymbol{dom}(F)$,

  then $[\![t_i]\!]^A\sigma' \downarrow$ and $([\![t_1]\!]^A\sigma', \ldots, [\![t_m]\!]^A\sigma') \in \boldsymbol{dom}(F)$,

  so that $[\![t]\!]^A\sigma$ and $[\![t]\!]^A\sigma'$ converge to the same value.

  Suppose for all $i \in \{1, \ldots, m\}$,

  $[\![t_i]\!]^A\sigma \downarrow$ and $([\![t_1]\!]^A\sigma, \ldots, [\![t_m]\!]^A\sigma) \notin \boldsymbol{dom}(F)$,

  or suppose there exists $i \in \{1, \ldots, m\}$, $[\![t_i]\!]^A\sigma \uparrow$,

  then both $[\![t]\!]^A\sigma \uparrow$ and $[\![t]\!]^A\sigma' \uparrow$.

- $t \equiv \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}$

  By i.h., since $\sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(t_i))$,

  so $[\![t_i]\!]^A\sigma \simeq [\![t_i]\!]^A\sigma', i = 1, 2$ and $[\![b]\!]^A\sigma \simeq [\![b]\!]^A\sigma'$,

  and the result follows directly from the definition.

$\square$

Corollary 3.3.6. If $\mathtt{x} \notin \boldsymbol{Var}(t)$ then $[\![t]\!]^A\sigma\{\mathtt{x}/a\} \simeq [\![t]\!]^A\sigma$.

Lemma 3.3.7 (Substitution lemma for terms).

$$[\![t]\!]^A\sigma\downarrow \quad \Rightarrow \quad ([\![t'[\mathtt{x}/t]]\!]^A\sigma \simeq [\![t']\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\})$$

Proof. Suppose $[\![t]\!]^A\sigma\downarrow$, we want to show:

$$[\![t'[\mathtt{x}/t]]\!]^A\sigma \simeq [\![t']\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}.$$

The proof is by structural induction on t'.

- $t' \equiv \mathtt{x}$

$LHS = [\![t'[\mathtt{x}/t]]\!]^A\sigma \simeq [\![t]\!]^A\sigma$

$RHS = [\![t']\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \simeq [\![t]\!]^A\sigma$ (by Definition 3.2.2)

- $t' \equiv \mathtt{y} \not\equiv \mathtt{x}$

$LHS = [\![t'[\mathtt{x}/t]]\!]^A\sigma \simeq [\![\mathtt{y}]\!]^A\sigma \simeq \sigma(\mathtt{y})$

$RHS = [\![\mathtt{y}]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$

$\simeq \sigma(\mathtt{y})$ (since $\mathtt{x} \notin \boldsymbol{Var}(t')$, by Corollary 3.3.6)

- $t' \equiv \mathsf{F}(t_1, \ldots, t_m)$

$$LHS = [\![\mathsf{F}(t_1, \ldots, t_m)[\mathtt{x}/t]]\!]^A\sigma$$

$$\simeq \mathsf{F}([\![t_1[\mathtt{x}/t]]\!]^A\sigma, \ldots, [\![t_1[\mathtt{x}/t]]\!]^A\sigma) \quad \text{(by Definitions 3.2.5, 3.3.2)}$$

$$RHS = [\![\mathsf{F}(t_1, \ldots, t_m)]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$$

$$\simeq \mathsf{F}([\![t_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}, \ldots, [\![t_m]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}) \quad \text{(by Definition 3.2.4)}$$

$$LHS \simeq RHS \quad \text{(by i.h.)}$$

- $t' \equiv \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}$

$$LHS = [\![(\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi})[\mathtt{x}/t]]\!]^A\sigma$$

$$\simeq [\![\mathsf{if}\ b[\mathtt{x}/t]\ \mathsf{then}\ t_1[\mathtt{x}/t]\ \mathsf{else}\ t_2[\mathtt{x}/t]\ \mathsf{fi}]\!]^A\sigma \quad \text{(by Definition 3.3.2)}$$

$$\simeq \begin{cases} [\![t_1[x/t]]\!]^A\sigma & \text{if } [\![b[x/t]]\!]^A\sigma \downarrow \mathtt{tt} \\ [\![t_2[x/t]]\!]^A\sigma & \text{if } [\![b[x/t]]\!]^A\sigma \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

$$RHS = [\![(\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi})]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$$

$$\simeq \begin{cases} [\![t_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} & \text{if } [\![b]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \downarrow \mathtt{tt} \\ [\![t_2]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} & \text{if } [\![b]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \downarrow \mathtt{ff} \\ \uparrow & \text{otherwise} \end{cases}$$

$$LHS \simeq RHS \quad \text{(by i.h.)}$$

□

Remark 3.3.8. (a) The functionality lemma (3.3.5) and substitution lemma(3.3.7) generalize the corresponding lemmas for total algebras [SA91, TZ00].

(b) The assumption $[\![t]\!]^A\sigma\downarrow$ in the substitution lemma is necessary to ensure that the variant state $\sigma\{x/[\![t]\!]^A\sigma\}$ is defined.

Chapter 4

The Proof system $\boldsymbol{PPL/SL}(A)$

We want to develop a predicate logic to support Hoare Logic for the programming language $\boldsymbol{SL}(\Sigma)$ over many-sorted partial algebras A. As our first approach, we use *partial predicate logic* $\boldsymbol{PPL}(\Sigma)$ to construct *partial or 3-valued* assertions. We then establish a mechanism for proving facts expressed in terms of partial assertions. In this approach, we have $\boldsymbol{Bool} \subset \boldsymbol{Assn}$, where $\boldsymbol{Bool}$ is the class of program terms of sort bool, and $\boldsymbol{Assn}$ is the class of assertions (§4.1). We will present the corresponding proof system $\boldsymbol{PPL/SL}(A)$ and prove its soundness. In the next chapter, we will present another approach (*total or 2-valued assertions*) for Hoare Logic over partial algebras.

4.1 3-valued assertion language

The class $\boldsymbol{Assn}$ of assertions $p, \ldots$ is defined by:

$$\begin{aligned} p \ ::= \ & \mathsf{true} \mid \mathsf{false} \mid \mathsf{not}\ p \mid p_1\ \mathsf{and}\ p_2 \mid p_1\ \mathsf{or}\ p_2 \\ & \mid \mathsf{F}(t_1, \ldots, t_n) \mid \mathsf{if}\ \ p\ \mathsf{then}\ \ p_1\ \mathsf{else}\ \ p_2\ \mathsf{fi}\ \ \mid \exists \mathtt{x}[p] \mid \forall \mathtt{x}[p] \end{aligned}$$

where $\mathsf{F} : s_1 \times \cdots \times s_m \to \mathsf{bool}$ and $t_i : s_i\ (i = 1, \ldots, n)$.

(Note that F could be the equality function eq_s at any equality sort s.)

We define a *partial evaluation function*, for $p \in \boldsymbol{Assn}$:

$$[\![p]\!]^A : \boldsymbol{State}(A) \stackrel{\cdot}{\to} \mathbb{B},$$

by structural induction on p. For the cases without quantifiers, the definition agrees with Definition 3.2.5 for program booleans. As with booleans, we have a (non-strict) conditional operator, and can hence define the operators cand and cor. As with program terms of sort bool, we can rewrite the logical operators as follows (*cf.* Remark 3.2.7):

$$[\![\mathsf{not}\ p_1]\!]^A\sigma \ \simeq \ \mathrm{not}([\![p_1]\!]^A\sigma)$$
$$[\![p_1\ \mathsf{and}\ p_2]\!]^A\sigma \ \simeq \ ([\![p_1]\!]^A\sigma \ \mathrm{and}\ [\![p_2]\!]^A\sigma)$$
$$[\![p_1\ \mathsf{or}\ p_2]\!]^A\sigma \ \simeq \ ([\![p_1]\!]^A\sigma \ \mathrm{or}\ [\![p_2]\!]^A\sigma)$$
$$[\![p_1\ \mathsf{cand}\ p_2]\!]^A\sigma \ \simeq \ ([\![p_1]\!]^A\sigma \ \mathrm{cand}\ [\![p_2]\!]^A\sigma)$$
$$[\![p_1\ \mathsf{cor}\ p_2]\!]^A\sigma \ \simeq \ ([\![p_1]\!]^A\sigma \ \mathrm{cor}\ [\![p_2]\!]^A\sigma)$$

For cases with quantifiers, we will use the definitions proposed by Kleene [Kle52] to define quantifiers for partial predicates:

$$[\![\exists \mathtt{x}[p]]\!]^A\sigma \;=\; \begin{cases} \mathsf{tt} & \text{if for some } \; a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/\; a\;\}\downarrow \mathsf{tt} \\ \mathsf{ff} & \text{if for all } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\downarrow \mathsf{ff} \\ \uparrow & \text{otherwise, } \textit{i.e.} \text{ if there is no } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\downarrow \mathsf{tt}, \\ & \text{but for some } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\uparrow \end{cases}$$

$$[\![\forall \mathtt{x}[p]]\!]^A\sigma \;=\; \begin{cases} \mathsf{tt} & \text{if for all } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\downarrow \mathsf{tt} \\ \mathsf{ff} & \text{if for some } \; a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\downarrow \mathsf{ff} \\ \uparrow & \text{otherwise, } \textit{i.e.} \text{ if for no } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\downarrow \mathsf{ff}, \\ & \text{but for some } a\in A_s \quad [\![P]\!]^A\sigma\{\mathtt{x}/a\}\uparrow. \end{cases}$$

4.2 Substitution for assertions

Definition 4.2.1 (Substitution for assertions). We define $p[\mathtt{x}/t]$ by structural induction on p (*cf.* [SA91]).

$\mathsf{true}[\mathtt{x}/t] \;\equiv\; \mathsf{true}$

$\mathsf{false}[\mathtt{x}/t] \;\equiv\; \mathsf{false}$

$(\mathsf{not}\ p)[\mathtt{x}/t] \;\equiv\; \mathrm{not}(p[\mathtt{x}/t])$

$(p_1\ \mathsf{and}\ p_2)[\mathtt{x}/t] \;\equiv\; p_1[\mathtt{x}/t]\ \mathrm{and}\ p_2[\mathtt{x}/t]$

$(p_1\ \mathsf{or}\ p_2)[\mathtt{x}/t] \;\equiv\; p_1[\mathtt{x}/t]\ \mathrm{or}\ p_2[\mathtt{x}/t]$

$\mathsf{F}(t_1,\ldots,t_m)[\mathtt{x}/t] \;\equiv\; \mathsf{F}(t_1[\mathtt{x}/t],\ldots,t_m[\mathtt{x}/t])$

$$(\textsf{if } p \textsf{ then } p_1 \textsf{ else } p_2 \textsf{ fi })[\mathtt{x}/t] \;\equiv\; \textsf{if } p[\mathtt{x}/t] \textsf{ then } p_1[\mathtt{x}/t] \textsf{ else } p_2[\mathtt{x}/t] \textsf{ fi}$$

$$\exists \mathtt{y}[p][\mathtt{x}/t] \;\equiv\; \begin{cases} \exists \mathtt{y}[p] & \text{if } \mathtt{y} \equiv \mathtt{x} \\ \exists \mathtt{y}[p[\mathtt{x}/t]] & \text{if } \mathtt{y} \not\equiv \mathtt{x} \text{ and } \mathtt{y} \notin \boldsymbol{Var}(t) \\ \exists \mathtt{y}_1[p[\mathtt{y}/\mathtt{y}_1][\mathtt{x}/t]] & \text{if } \mathtt{y} \not\equiv \mathtt{x} \text{ and } \mathtt{y} \in \boldsymbol{Var}(t) \\ & \text{where } \mathtt{y}_1 \text{ is the first variable with sort same as } t \\ & \text{such that } \mathtt{y}_1 \not\equiv \mathtt{x} \text{ and } \mathtt{y}_1 \notin \boldsymbol{Var}(p,t) \end{cases}$$

$$\forall \mathtt{y}[p][\mathtt{x}/t] \;\equiv\; \begin{cases} \forall \mathtt{y}[p] & \text{if } \mathtt{y} \equiv \mathtt{x} \\ \forall \mathtt{y}[p[\mathtt{x}/t]] & \text{if } \mathtt{y} \not\equiv \mathtt{x} \text{ and } \mathtt{y} \notin \boldsymbol{Var}(t) \\ \forall \mathtt{y}_1[p[\mathtt{y}/\mathtt{y}_1][\mathtt{x}/t]] & \text{if } \mathtt{y} \not\equiv \mathtt{x} \text{ and } \mathtt{y} \in \boldsymbol{Var}(t) \\ & \text{where } \mathtt{y}_1 \text{ is the first variable with sort same as } t \\ & \text{such that } \mathtt{y}_1 \not\equiv \mathtt{x} \text{ and } \mathtt{y}_1 \notin \boldsymbol{Var}(p,t) \end{cases}$$

4.3 Functionality and substitution lemmas for assertions

(Compare §3.3)

Lemma 4.3.1 (Functionality lemma for assertions).

$$\text{If} \quad \sigma \approx \sigma'(\textsf{rel } \boldsymbol{Var}(p)) \quad \text{then} \quad [\![p]\!]^A\sigma \;\simeq\; [\![p]\!]^A\sigma'.$$

Proof. Suppose $\sigma \approx \sigma'(\textsf{rel } \boldsymbol{Var}(p))$. We will prove $[\![p]\!]^A\sigma \;\simeq\; [\![p]\!]^A\sigma'$ by structural induction on p.

- $p \equiv \mathsf{true}$

$[\![\mathsf{true}]\!]^A\sigma \; = \; \mathtt{tt} \; = \; [\![\mathsf{true}]\!]^A\sigma'$

- $p \equiv \mathsf{false}$

$[\![\mathsf{false}]\!]^A\sigma \; = \; \mathtt{ff} \; = \; [\![\mathsf{false}]\!]^A\sigma'$

- $p \equiv \mathsf{not}\ p_1$

$LHS \; \equiv \; [\![\mathsf{not}\ p_1]\!]^A\sigma \; \simeq \; \text{not } ([\![p_1]\!]^A\sigma) \;$ (by definition)

$RHS \; \equiv \; [\![\mathsf{not}\ p_1]\!]^A\sigma' \; \simeq \; \text{not } ([\![p_1]\!]^A\sigma') \;$ (by definition)

By i.h., $[\![p_1]\!]^A\sigma \; \simeq \; [\![p_1]\!]^A\sigma'$

- $p \; \equiv \; p_1 \ \mathsf{and}\ p_2$

$$\begin{aligned} LHS \; &= \; [\![p_1 \ \mathsf{and}\ p_2]\!]^A\sigma \\ &\simeq \; [\![p_1]\!]^A\sigma \ \text{ and } \ [\![p_2]\!]^A\sigma \quad \text{(by definition)} \\ &\simeq \; [\![p_1]\!]^A\sigma' \ \text{ and } \ [\![p_2]\!]^A\sigma' \quad \text{(by i.h.)} \\ &\simeq \; [\![p_1 \ \mathsf{and}\ p_2]\!]^A\sigma' \\ &= \; RHS \text{ (by definition)} \end{aligned}$$

- $p \; \equiv \; p_1 \ \mathsf{or}\ p_2$

$$\begin{aligned} LHS \; &= \; [\![p_1 \ \mathsf{or}\ p_2]\!]^A\sigma \\ &\simeq \; [\![p_1]\!]^A\sigma \ \text{ or } [\![p_2]\!]^A\sigma \quad \text{(by definition)} \\ &\simeq \; [\![p_1]\!]^A\sigma' \ \text{ or } \ [\![p_2]\!]^A\sigma' \quad \text{(by i.h.)} \\ &\simeq \; [\![p_1 \ \mathsf{or}\ p_2]\!]^A\sigma' \\ &= \; RHS \text{ (by definition)} \end{aligned}$$

- $p \; \equiv \; \mathsf{F}(t_1, \ldots, t_m)$

Immediately from the functionality lemma for terms (3.3.5).

- $p \; \equiv \mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2\ \mathsf{fi}$

By assumption, $\sigma \; \approx \; \sigma'(\mathsf{rel}\ \boldsymbol{Var}(p_i))$, for $i = 0, 1, 2$

So by i.h., $[\![p_i]\!]^A\sigma \;\simeq\; [\![p_i]\!]^A\sigma'$, for $i = 0, 1, 2$

and the result follows directly from the definition.

- $p \;\equiv\; \exists \mathtt{x}[p_1]$

By assumption, $\sigma \approx \sigma'(\textsf{rel } \boldsymbol{Var}(p))$,

and also $\sigma \approx \sigma'(\textsf{rel } \boldsymbol{Var}(p_1)\backslash\{\mathtt{x}\})$.

Hence for all a, $\sigma\{\mathtt{x}/a\} \restriction \boldsymbol{Var}(p_1) \;\approx\; \sigma'\{\mathtt{x}/a\} \restriction \boldsymbol{Var}(p_1)$.

By i.h., $[\![p_1]\!]^A\sigma\{\mathtt{x}/a\} \;\simeq\; [\![p_1]\!]^A\sigma'\{\mathtt{x}/a\}$

and there exist a $[\![p_1]\!]^A\sigma\{\mathtt{x}/a\}$ iff there exists a $[\![p_1]\!]^A\sigma'\{\mathtt{x}/a\}$.

i.e. $[\![p]\!]^A\sigma \;\simeq\; [\![p]\!]^A\sigma'$.

- $p \;\equiv\; \forall \mathtt{x}[p_1]$

Similarly.

□

Remark 4.3.2. This generalizes the functionality (or "coincidence") lemma for total algebras [SA91, TZ00].

Corollary 4.3.3. If $\mathtt{x} \notin \boldsymbol{Var}(p)$ then $[\![p]\!]^A\sigma\{\mathtt{x}/a\} \;\simeq\; [\![p]\!]^A\sigma$.

Lemma 4.3.4 (Substitution lemma for assertions).

$$[\![t]\!]^A\sigma\downarrow \;\Rightarrow\; ([\![p[\mathtt{x}/t]]\!]^A\sigma \;\simeq\; [\![p]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\})$$

Proof. Take an arbitrary σ and assume $[\![t]\!]^A\sigma\downarrow$. We want to prove that

$$([\![p[\mathtt{x}/t]]\!]^A\sigma \;\simeq\; [\![p]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\})\downarrow \mathtt{tt}$$

by structural induction on p.

- $p \equiv \mathsf{true}$

$[\![\mathsf{true}]\!]^A\sigma \ = \ \mathtt{tt} \ = \ [\![\mathsf{true}]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}.$

- $p \equiv \mathsf{false}$

$[\![\mathsf{false}]\!]^A\sigma \ = \ \mathtt{ff} \ = \ [\![\mathsf{false}]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}.$

- $p \equiv \mathsf{not}\ p_1$

$$\begin{aligned}
\text{LHS} &= [\![(\mathsf{not}\ p_1)[\mathtt{x}/t]]\!]^A\sigma \\
&\simeq [\![\mathsf{not}\ (p_1[\mathtt{x}/t])]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq \text{not } [\![p_1[\mathtt{x}/t]]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq \text{not } [\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \quad \text{(by i.h.)} \\
&\simeq [\![\mathsf{not}\ p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \\
&= \text{RHS} \quad \text{(by definition).}
\end{aligned}$$

- $p \ \equiv \ p_1 \ \mathsf{and} \ p_2$

$$\begin{aligned}
\text{LHS} &= [\![(p_1 \ \mathsf{and} \ p_2)[\mathtt{x}/t]]\!]^A\sigma \\
&\simeq [\![(p_1[\mathtt{x}/t] \text{ and } p_2[\mathtt{x}/t])]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq [\![p_1[\mathtt{x}/t]]\!]^A\sigma \ \text{ and } \ [\![p_2[\mathtt{x}/t]]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq [\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \text{ and } [\![p_2]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \quad \text{(by i.h.)} \\
&\simeq [\![p_1 \ \mathsf{and} \ p_2]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \quad \text{(by definition)} \\
&= \text{RHS.}
\end{aligned}$$

- $p \ \equiv \ p_1 \ \mathsf{or} \ p_2$

$$\begin{aligned}
\text{LHS} &= [\![(p_1 \ \mathsf{or} \ p_2)[\mathtt{x}/t]]\!]^A\sigma \\
&\simeq [\![(p_1[\mathtt{x}/t] \ \text{ or } p_2[\mathtt{x}/t])]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq [\![p_1[\mathtt{x}/t]]\!]^A\sigma \ \text{ or } \ [\![p_2[\mathtt{x}/t]]\!]^A\sigma \quad \text{(by definition)} \\
&\simeq [\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \ \text{ or } \ [\![p_2]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \quad \text{(by i.h.)} \\
&\simeq [\![p_1 \ \mathsf{or} \ p_2]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \quad \text{(by definition)}
\end{aligned}$$

$=$ RHS.

- $p \;\equiv\; \mathsf{F}(t_1,\ldots,t_m)$

Immediately from the substitution lemma for terms (Lemma 3.3.7).

- $p \;\equiv \mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2\ \mathsf{fi}$

As with the substitution lemma for terms (Lemma 3.3.7).

- $p \;\equiv\; \exists \mathsf{y}[p_1]$

Subcase 1 : $\mathtt{x} \equiv \mathtt{y}$

$[\![\exists \mathtt{x}[p_1][\mathtt{x}/t]]\!]^A\sigma$

$\simeq\; [\![\exists \mathtt{x}[p_1]]\!]^A\sigma$ (by definition)

$\simeq\; [\![\exists \mathtt{x}[p_1]]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$ (by Corollary 4.3.3)

Subcase 2 : $\mathtt{x} \not\equiv \mathtt{y}$ and $\mathtt{y} \notin \boldsymbol{Var}(t)$

We only prove the case $[\![\exists \mathsf{y}[p_1][\mathtt{x}/t]]\!]^A\sigma \downarrow \mathtt{tt}$.

(The case $\mathtt{ff}$ is similar, and the case $\uparrow$ is easy.)

$[\![\exists \mathsf{y}[p_1][\mathtt{x}/t]]\!]^A\sigma$

$\simeq\; [\![\exists \mathsf{y}[p_1[\mathtt{x}/t]]]\!]^A\sigma$ (by definition)

$\simeq\; [\![p_1[\mathtt{x}/t]]\!]^A\sigma\{\mathtt{y}/\ a\}$ (by definition with a suitably chosen)

$\simeq\; [\![p_1]\!]^A\sigma\{\mathtt{y}/a\}\{\mathtt{x}/[\![t]\!]^A\sigma\{\mathtt{y}/\ a\}\}$ (by i.h.)

$\simeq\; [\![p_1]\!]^A\sigma\{\mathtt{y}/\ a\}\{\mathtt{x}/[\![t]\!]^A\sigma\}$ (since $\mathtt{y} \notin \boldsymbol{Var}(t)$)

$\simeq\; [\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}\{\mathtt{y}/\ a\}$ (Lemma 3.2.3 for variant of state)

$\simeq\; [\![\exists \mathsf{y}[p_1]]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$ (by definition)

Subcase 3 : $\mathtt{x} \not\equiv \mathtt{y}$ and $\mathtt{y} \in \boldsymbol{Var}(t)$

We only prove the case $[\![\exists \mathsf{y}[p_1][\mathtt{x}/t]]\!]^A\sigma \downarrow \mathtt{tt}$.

(The case $\mathtt{ff}$ is similiar, and the case $\uparrow$ is easy.)

$[\![\exists \mathsf{y}[p_1][\mathtt{x}/t]]\!]^A\sigma$

$\simeq$ $[\![\exists \mathtt{y}'[p_1[\mathtt{y}/\mathtt{y}'][\mathtt{x}/t]]]\!]^A\sigma$ (by definition)

$\simeq$ $[\![p_1[\mathtt{y}/\mathtt{y}'][\mathtt{x}/t]]\!]^A\sigma\{\mathtt{y}'/\ a\}$ (by definition with a suitably chosen)

$\simeq$ $[\![p_1[\mathtt{y}/\mathtt{y}']]\!]^A(\sigma\{\mathtt{y}'/\ a\}\{\mathtt{x}/[\![t]\!]^A\sigma\})$ (by i.h.)

$\simeq$ $[\![p_1]\!]^A\sigma\{\mathtt{y}'/\ a\}\{\mathtt{x}/[\![t]\!]^A\sigma\}\{\mathtt{y}/\sigma\{\mathtt{y}'/\ a\}\{\mathtt{x}/[\![t]\!]^A\sigma\}(\mathtt{y}')\}$ (by i.h.)

$\simeq$ $[\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}\{\mathtt{y}/\ a\}\{\mathtt{y}'/\ a\}$ (Lemma 3.2.3 for variant of state)

$\simeq$ $[\![p_1]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}\{\mathtt{y}/\ a\}$ (since $\mathtt{y}' \notin \boldsymbol{Var}(p_1)$)

$\simeq$ $[\![\exists \mathtt{y}[p_1]]\!]^A\sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$ (by definition)

$\square$

Remark 4.3.5. This generalizes the substitution lemma for total algebras [SA91, TZ88].

4.4 Hoare formulae

Definition 4.4.1 (Syntax of Hoare formulae). The class of Hoare formulae $f, \dots$ is defined by:

$$f ::= p \mapsto q \mid \{p\}\ S\ \{q\}$$

Definition 4.4.2 (A-validity of Hoare formulae). Validity of a formula f w.r.t. A, or A-validity of f, written as

$$\models_A f$$

is defined as:

$Case1: \ f \ \equiv \ p \ \mapsto \ q$

$\models_A f$ iff for all σ, if $[\![p]\!]^A\sigma \downarrow \mathtt{tt}$ then $[\![q]\!]^A\sigma \downarrow \mathtt{tt}$

$Case2: \ f \ \equiv \ \{\ p\ \}S\ \{q\}$

$\models_A f$ iff for all σ, if $[\![p]\!]^A\sigma \downarrow \mathtt{tt}$ then $[\![S]\!]^A\sigma \downarrow$ and $[\![q]\!]^A([\![S]\!]^A\sigma) \downarrow \mathtt{tt}$

Remark 4.4.3. This notion of validity is also called *total correctness* of Hoare formulae as opposed to *partial correctness* (*cf.* §1.1).

4.5 The proof system $\boldsymbol{PPL/SL}(A)$

Assumption 4.5.1 (Definedness assumption). In working with $\boldsymbol{PPL}(\Sigma)$, we must assume the algebra A satisfies :

For all Σ-sorts s, we can define an assertion $\textsf{def}_s(t)$ such that for all Σ-terms t and $\sigma \in \boldsymbol{State}(A)$,

$$[\![\textsf{def}_s(t)]\!]^A\sigma \downarrow \quad \Leftrightarrow \quad [\![t]\!]^A\sigma \downarrow .$$

For example, for every *total equality* or *semi-equality* Σ-sort s *e.g.* $s = \mathsf{nat}$, we can take

$$\mathsf{def}_s(t) \quad \equiv_{df} \quad \mathsf{eq}_s(t,t).$$

For booleans b, take

$$[\![\mathsf{def}_{\mathsf{bool}}(b)]\!]^A \quad \equiv_{df} \quad (b \ \mathsf{or} \ \mathsf{not}(b))$$

and for the sort real, we can take

$$\mathsf{def}_{\mathsf{real}}(t) \quad \equiv_{df} \quad \mathsf{less}_{\mathsf{real}}(t, t+1).$$

Lemma 4.5.2 (Definedness for *PPL*(Σ)).

$$\models_A \quad \mathsf{def}_{\mathsf{bool}}(b) \qquad \Leftrightarrow \qquad \text{for all } \sigma, [\![b]\!]^A \sigma \downarrow (\mathsf{tt} \text{ or } \mathsf{ff}).$$

Proof. Immediate from the definition. □

Definition 4.5.3 (Inferences and their validity). An *inference* is a construct of the form:

$$\frac{f_1, \ldots, f_n}{f} \tag{1}$$

with $f_i, (i = 1, \ldots, n)$ and $f \in \boldsymbol{Form}$.

The inference (1) above is called *A-valid* whenever

$$\models_A f_i \ (i = 1, \ldots, n) \quad \Rightarrow \quad \models_A \ f$$

i.e. A-validity of the premisses implies A-validity of the conclusion.

Note that an *axiom* is an inference without premisses (*i.e.* $n = 0$).

Definition 4.5.4 (Proof system *PPL/SL*(A)). The proof system ***PPL/SL***(A) consists of the following axioms and inference rules:

(*a*) *Axioms*:

1. (*Assertions*)

$$\text{All } A\text{-valid formulae} \quad p \mapsto q$$

2. (*Assignment*)

$$\{\mathsf{def}_s(t) \ \mathsf{cand}\ p[\mathtt{x}/t]\} \quad \mathtt{x} := t \quad \{p\}$$

(*b*) *Inference rules*:

1. (*Consequence*)

$$\frac{p \mapsto p_1, \quad \{p_1\}S\{q_1\}, \quad q_1 \mapsto q}{\{p\}S\{q\}}$$

2. (*Composition*)

$$\frac{\{p\}S_1\{q\}, \quad \{q\}S_2\{r\}}{\{p\}S_1; S_2\{r\}}$$

3. (*Conditional*)

$$\frac{p \mapsto \mathsf{def}_{\mathsf{bool}}(b), \quad \{p \ \mathsf{cand}\ b\}S_1\{q\}, \quad \{p \ \mathsf{cand}\ (\mathsf{not}\ b)\}S_2\{q\}}{\{p\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\ \{q\}}$$

Remark 4.5.5.

PPL/SL(A) is not a "proof system" in the normal sense, since for any N-standard algebra A, A-validity of assertions is not decidable, or even semi-decidable (by Tarski's Theorem [Tar55] that arithmetical truth is not arithmetically definable), and so (from

Axiom 1) the axioms of $\boldsymbol{PPL/SL}(A)$ are not decidable. This remark also applies to the other proof systems considered in this thesis, and to proof systems for Hoare Logic in general [Coo78].

4.6 Soundness of proof system $\boldsymbol{PPL/SL}(A)$

Lemma 4.6.1 (A-validity of assignment rule).

$$\models_A \quad \{\mathsf{def}_s(t) \ \mathsf{cand}\ p[\mathtt{x}/t]\}\ \mathtt{x} := t \ \ \{p\}$$

Proof. Assume for all σ,

$$[\![\mathsf{def}_s(t)\ \mathsf{cand}\ \ p[x/t]]\!]^A\sigma \downarrow \mathtt{tt}. \tag{4.1}$$

We want to show that

$$[\![x := t]\!]^A\sigma \downarrow \tag{4.2}$$

and

$$[\![p]\!]^A([\![x := t]\!]^A\sigma) \downarrow \mathtt{tt} \tag{4.3}$$

By (4.1):

$$[\![\mathsf{def}(t)]\!]^A\sigma \downarrow \mathtt{tt} \tag{4.4}$$

and

$$[\![p[x/t]]\!]^A\sigma \downarrow \mathtt{tt} \tag{4.5}$$

By (4.4) and the definedness lemma (4.5.2),

$$[\![t]\!]^A\sigma \downarrow \tag{4.6}$$

Hence by Definition 3.3.1,

$$[\![x := t]\!]^A\sigma \downarrow \sigma\{x/[\![t]\!]^A\sigma\}. \tag{4.7}$$

By (4.6) and the substitution lemma for assertions (4.3.4),

$$[\![p[x/t]]\!]^A\sigma \quad \simeq \quad [\![p]\!]^A\sigma\{x/[\![t]\!]^A\sigma\} \tag{4.8}$$

So

$$\begin{aligned} [\![p]\!]^A([\![x := t]\!]^A\sigma) &\simeq [\![p]\!]^A\sigma\{x/[\![t]\!]^A\sigma\} \ \text{ by (4.7)} \\ &\simeq [\![p[x/t]]\!]^A\sigma \ \text{ by (4.8)} \\ &\downarrow \mathtt{tt} \ \text{ by (4.5)} \end{aligned}$$

i.e. (4.2) and (4.3) holds. □

Lemma 4.6.2 (*A*-validity of consequence rule). The inference

$$\frac{(p \mapsto p_1), \quad \{p_1\}S\{q_1\}, \quad (q_1 \mapsto q)}{\{p\}S\{q\}}$$

is A-valid.

Proof. Assume that for all σ,

$$\models_A \quad p \mapsto p_1 \tag{4.9}$$

and

$$\models_A \quad \{p_1\}S\{q_1\} \tag{4.10}$$

and

$$\models_A \quad q_1 \mapsto q \tag{4.11}$$

We must show

$$\models_A \quad \{p\}S\{q\} \tag{4.12}$$

Take any σ and assume $[\![p]\!]^A\sigma \downarrow \mathtt{tt}$. By (4.9)

$$[\![p_1]\!]^A\sigma \downarrow \mathtt{tt},$$

by (4.10)

$$[\![S]\!]^A\sigma\downarrow \text{ and } [\![q_1]\!]^A([\![S]\!]^A\sigma)\downarrow \mathtt{tt},$$

and by (4.11),

$$[\![q]\!]^A([\![S]\!]^A\sigma)\downarrow \mathtt{tt}.$$

i.e. (4.12) holds. □

Lemma 4.6.3 (*A*-validity of sequential composition rule).

The inference

$$\frac{\{p\}S_1\{q\},\quad \{q\}S_2\{r\}}{\{p\}S_1;S_2\{r\}}$$

is A-valid.

Proof. Assume that for all σ,

$$\models_A \quad \{p\}S_1\{q\} \tag{4.13}$$

and

$$\models_A \quad \{q\}S_2\{r\} \tag{4.14}$$

We must show

$$\models_A \quad \{p\}S_1; S_2\{r\}. \tag{4.15}$$

Take any σ and assume $[\![p]\!]^A\sigma \downarrow \mathsf{tt}$. By (4.13)

$$[\![S_1]\!]^A\sigma \downarrow \text{ and } [\![q]\!]^A([\![S_1]\!]^A\sigma) \downarrow \mathsf{tt}. \tag{4.16}$$

By (4.14), for $\sigma' = [\![S_1]\!]^A\sigma$, we infer that

$$[\![S_2]\!]^A\sigma' \downarrow \text{ and } [\![r]\!]^A([\![S_2]\!]^A\sigma') \downarrow \mathsf{tt}, \tag{4.17}$$

and so by (4.16) and (4.17), and the semantics of composition,

$$[\![S_1; S_2]\!]^A\sigma \downarrow \text{ and } [\![r]\!]^A([\![S_1; S_2]\!]^A\sigma) \downarrow \mathsf{tt}.$$

□

Lemma 4.6.4 (*A*-validity of conditional rule).

The inference

$$\frac{p \;\mapsto \mathsf{def}_{\mathsf{bool}}(b)\ ,\quad \{p \;\mathsf{cand}\; b\}S_1\{q\},\quad \{p \;\mathsf{cand}\; (\mathsf{not}\; b)\}S_2\{q\}}{\{p\}\;\mathsf{if}\; b \;\mathsf{then}\;\; S_1 \;\mathsf{else}\;\; S_2 \;\mathsf{fi}\; \{q\}}$$

is A-valid.

Proof. Assume that for all σ,

$$\models_A \quad p \;\mapsto \mathsf{def}_{\mathsf{bool}}(b) \tag{4.18}$$

and

$$\models_A \quad \{p \;\mathsf{cand}\; b\}S_1\{q\} \tag{4.19}$$

and

$$\models_A \quad \{p \;\mathsf{cand}\; (\mathsf{not}\; b)\}S_2\{q\}. \tag{4.20}$$

We must show

$$\models_A \quad \{p\}\;\mathsf{if}\; b\;\mathsf{then}\;\; S_1\;\mathsf{else}\;\; S_2\;\mathsf{fi}\;\{q\} \tag{4.21}$$

Take any σ and assume

$$[\![p]\!]^A\sigma \downarrow \mathtt{tt}. \tag{4.22}$$

By (4.18) and the definedness lemma (Lemma 4.5.2),

$$[\![b]\!]^A\sigma \downarrow (\mathtt{tt} \text{ or } \mathtt{ff}). \tag{4.23}$$

Case $1 : [\![b]\!]^A\sigma \downarrow \mathtt{tt}$. By (4.19) and (4.22)

$$[\![S_1]\!]^A\sigma\downarrow \text{ and } [\![q]\!]^A([\![S_1]\!]^A\sigma)\downarrow \mathtt{tt}. \tag{4.24}$$

Case 2 : $[\![b]\!]^A\sigma\downarrow \mathtt{ff}$. By (4.20) and (4.22)

$$[\![S_2]\!]^A\sigma\downarrow \text{ and } [\![q]\!]^A([\![S_2]\!]^A\sigma)\downarrow \mathtt{tt}. \tag{4.25}$$

Then (4.21) follows from (4.22), (4.23), (4.24) and (4.25) . □

Theorem 4.6.5 (Soundness of $\boldsymbol{PPL/SL}(A)$). For all $f \in \boldsymbol{Form}$,

$$\boldsymbol{PPL/SL}(A) \vdash f \quad \Rightarrow \quad \models_A f$$

Proof. Since the axioms are A-valid and the proof rules preserve A-validity (Lemmas 4.6.1 – –4.6.4), the result follows by induction on the proof length. □

4.7 The weakest precondition

We are often interested, not just in any precondition p such that, for given S and q,

$$\models_A \{ p \} S \{ q \},$$

but in a "best possible" precondition p. This idea is made precise in the concept of *weakest precondition.* The reasons for studying the *weakest precondition* are:

(1) it can be helpful in proving the completeness of proof systems (which we will not actually do in this thesis); and

(2) we will use the weakest precondition in our case study (Chapter 8) to find intermediate assertions for the correctness proof.

Definition 4.7.1 (Weakest precondition). Given a statement S and predicate q on A, a *weakest precondition* with respect to S and q is an assertion, written $\mathsf{wp}_A(S, q)$, which holds at any $\sigma \in \boldsymbol{State}(A)$ (*i.e.* its value at σ is $\mathtt{tt}$) iff

$$[\![S]\!]^A\sigma\downarrow \quad \text{and} \quad [\![q]\!]^A([\![S]\!]^A)\sigma\downarrow \mathtt{tt}$$

Definition 4.7.2 (weak A-implication and weak A-equivalence).

(a) $p \underset{A}{\models\!\!\!\Rightarrow} q$ iff for all σ $([\![p]\!]^A\sigma\downarrow \mathtt{tt} \Rightarrow [\![q]\!]^A\sigma\downarrow \mathtt{tt})$

(b) $p \underset{A}{\models\!\!\!\dashv} q$ iff $p \underset{A}{\models\!\!\!\Rightarrow} q$ and $q \underset{A}{\models\!\!\!\Rightarrow} p$

i.e. for all σ $([\![p]\!]^A\sigma\downarrow \mathtt{tt} \Leftrightarrow [\![q]\!]^A\sigma\downarrow \mathtt{tt})$.

Remark 4.7.3.

$$p \underset{A}{\models\!\!\!\Rightarrow} q \quad \text{iff} \quad \models_A (p \mapsto q)$$

Definition 4.7.4 (Strong A-equivalence).

(a) p and q are *strongly A-equivalent* iff for all $\sigma \in \boldsymbol{State}(A)$, all three of the following hold:

$$(1) \quad [\![p]\!]^A\sigma\downarrow \mathtt{tt} \quad \Leftrightarrow \quad [\![q]\!]^A\sigma\downarrow \mathtt{tt},$$

$$(2) \quad [\![p]\!]^A\sigma\downarrow \mathtt{ff} \quad \Leftrightarrow \quad [\![q]\!]^A\sigma\downarrow \mathtt{ff},$$

$$(3) \quad [\![p]\!]^A\sigma\uparrow \quad \Leftrightarrow \quad [\![q]\!]^A\sigma\uparrow .$$

(b) p and q are *weakly A-equivalent* iff (1) alone holds.

The weakest precondition $\mathsf{wp}_A(S,q)$ is characterized by the following two properties :

Lemma 4.7.5. For all $q \in \boldsymbol{Assn}$,

(a) $\models_A \; \{ \; \mathsf{wp}_A(S,q) \; \} \; S \; \{ \; q \; \}$

(b) For each r, if $\models_A \; \{ \; r \; \} \; S \; \{ \; q \; \}$, then $r \underset{A}{\models\!\!\Rightarrow} \mathsf{wp}_A(S,q)$

Proof. (a) We have to show that for all σ,

$$[\![\mathsf{wp}_A(S,q)]\!]^A\sigma \downarrow \mathsf{tt} \quad \Rightarrow \quad [\![S]\!]^A\sigma \downarrow \text{and } [\![q]\!]^A\sigma \downarrow \mathsf{tt},$$

which follows immediately from Definition 4.7.1.

(b) Choose some r and assume that for all σ,

$$\begin{aligned} [\![r]\!]^A\sigma \downarrow \mathsf{tt} \quad & \Rightarrow \quad [\![S]\!]^A\sigma \downarrow \text{and } [\![q]\!]^A\sigma \downarrow \mathsf{tt} \\ & \Rightarrow \quad [\![\mathsf{wp}_A(S,q)]\!]^A\sigma \downarrow \mathsf{tt} \qquad \text{(by Definition 4.7.1).} \end{aligned}$$

Hence

$$r \underset{A}{\models\!\!\Rightarrow} \mathsf{wp}_A(S,q).$$

□

Corollary 4.7.6 (Uniqueness of weakest precondition). Any two weakest preconditions of S and q on A are weakly A-equivalent.

Proof. Clear from the definition. □

We can therefore speak of "the" weakest precondition of S and q on A, since it is unique up to weak A-equivalence.

Now we show that the preconditions in our proof rules are the best possible, *i.e.*, the weakest.

Lemma 4.7.7. For each $q \in \boldsymbol{Assn}$,

$$\mathsf{wp}_A(\mathtt{x} := t, q) \underset{A}{\models\!\!\!\dashv} \mathsf{def}_s(t) \;\; \mathsf{cand}\; q[\mathtt{x}/t]$$

Proof. This follows directly from Definition 4.7.1 and Lemma 4.3.4. □

Remark 4.7.8 (Strict and non-strict logical operator). It is easy to check that

$$p \;\mathsf{cand}\; q \quad \underset{A}{\models\!\!\!\dashv} \quad p \;\mathsf{and}\; q$$

However they are *not* strongly equivalent. But since they are weakly equivalent, we could have defined

$$\mathsf{wp}_A(x := t, q) \quad =_{df} \quad \mathsf{def}_s(t) \;\mathsf{and}\; p[\mathtt{x}/t].$$

The question then is: why did we choose to define the weakest precondition using the non-strict logical operator 'cand' instead of the strict logical operator 'and' ? The answer is that this seem to reflect the meaning of the assertions more clearly. Note incidently that 'cor' is *not* (even) weakly equivalent to 'or', in fact,

$$(p \ \textsf{or} \ \ q) \ \underset{A}{\models\!\!\!\Rightarrow} \ (p \ \textsf{cor} \ \ q)$$

but (in general)

$$(p \ \textsf{cor} \ \ q) \ \underset{A}{\not\models\!\!\!\Rightarrow} \ (p \ \textsf{or} \ \ q)$$

Lemma 4.7.9. For each $q \in \boldsymbol{Assn}$,

(a) $\textsf{wp}_A(S_1; S_2, q) \ \underset{A}{\models\!\dashv} \ \textsf{wp}_A(S_1, (\textsf{wp}_A(S_2, q))$

(b) $\textsf{wp}_A(\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, q) \ \underset{A}{\models\!\dashv} \ \textsf{if } b \textsf{ then } \ \textsf{wp}_A(S_1, q) \textsf{ else } \textsf{wp}_A(S_2, q) \textsf{ fi}$

Proof. (a) For each σ,

$[\![\textsf{wp}_A(S_1; S_2, q)]\!]^A \sigma \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![S_1; S_2]\!]^A \sigma \downarrow$ and $[\![q]\!]^A([\![S_1; S_2]\!]^A \sigma) \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![S_2]\!]^A([\![S_1]\!]^A \sigma) \downarrow$ and $[\![q]\!]^A([\![S_2]\!]^A([\![S_1]\!]^A \sigma)) \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![S_1]\!]^A \sigma \downarrow$ and $[\![\textsf{wp}_A(S_2, q)]\!]^A([\![S_1]\!]^A \sigma) \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![\textsf{wp}_A(S_1, \textsf{wp}_A(S_2, q))]\!]^A \sigma \downarrow \mathsf{tt}$

(b) For each σ,

$[\![\textsf{wp}_A(\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, q)]\!]^A \sigma \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}]\!]^A \sigma \downarrow$ and $[\![q]\!]^A([\![\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}]\!]^A \sigma) \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![b]\!]^A \sigma \downarrow \mathsf{tt}$ and $[\![q]\!]^A([\![S_1]\!]^A \sigma) \downarrow \mathsf{tt}$ or $[\![b]\!]^A \sigma \downarrow \mathsf{ff}$ and $[\![q]\!]^A([\![S_2]\!]^A \sigma) \downarrow \mathsf{tt}$

$\Leftrightarrow \quad [\![\textsf{if } b \textsf{ then } \textsf{wp}_A(S_1, q) \textsf{ else } \textsf{wp}_A(S_2, q) \textsf{ fi}]\!]^A \sigma \downarrow \mathsf{tt}$. $\square$

Chapter 5

The Proof System $\boldsymbol{TPL}/\boldsymbol{SL}(A)$

As our second approach to a predicate logic underlying Hoare Logic over partial algebras, we will define *total* or *2-valued* predicate logic $\boldsymbol{TPL}(\Sigma)$ and use it to construct *total* assertions. In this approach, we have $\boldsymbol{Bool} \bigcap \boldsymbol{Assn} = \emptyset$ (compare with $\boldsymbol{Bool} \subset \boldsymbol{Assn}$ in $\boldsymbol{PPL}(\Sigma)$ in Chapter 4). We will then present the second proof system $\boldsymbol{TPL}/\boldsymbol{SL}(A)$ and prove its soundness.

5.1 Total predicate logic

The main objective of $\boldsymbol{TPL}(\Sigma)$ is to remain within classical predicate logic, hence it is built on *classical 2-valued total predicate logic with equality.* $\boldsymbol{TPL}(\Sigma)$ is constructed in two steps. The first step gives the interpretation of the atomic assertions. In the second step, assertions are built from *atomic assertions* by the classical 2-valued logical connectives. More specifically, $\boldsymbol{TPL}(\Sigma)$ is defined in such a way that :

(i) If we evaluate an *atomic assertion* $(t_1 = t_2)$ at a state where either t_1 or t_2 is undefined, then this atomic assertion will evaluate to ff.

(ii) Compound assertions are then evaluated as in classical (total) predicate logic.

This approach to predicate logic for partially defined terms has been defined in [Far90, Par93].

5.2 2-valued assertion language

(Compare §4.1.) *Atomic assertions* have the form (only)

$$t_1^s \ = \ t_2^s$$

i.e. total equality between two Σ-terms of sort s, for all Σ-sorts S (not just equality sorts). Note that '=' is *not* the same as the boolean equality operator eq_s, which exists only at equality sorts s, and may be partial.

The class $\boldsymbol{Assn}(\Sigma)$ of assertions $p, q, \ldots$ is defined by :

$$p \ ::= (t_1^s \ = \ t_2^s) \mid p_1 \ \wedge \ p_2 \mid p_1 \ \vee \ p_2 \mid \ \neg p \mid \exists x[p] \mid \forall x[p]$$

where s is any Σ-sort. For $p \in \boldsymbol{Assn}(\Sigma)$, we define the semantic function

$$[\![p]\!]^A : \boldsymbol{State}(A) \ \to \mathbb{B}$$

where $[\![p]\!]^A\sigma$ is the value of p in A at state σ. Note this function is a *total* 2-valued function. The definition is by structural induction on p :

$$[\![t_1 \ = \ t_2]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if } [\![t_1]\!]^A\sigma \downarrow a \ \text{ and } \ [\![t_2]\!]^A\sigma \downarrow a \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$[\![\neg p]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if } [\![p]\!]^A\sigma = \mathtt{ff} \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$[\![p_1 \ \wedge \ p_2]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if } [\![p_1]\!]^A\sigma = \mathtt{tt} \ \text{ and } \ [\![p_2]\!]^A\sigma = \mathtt{tt} \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$[\![p_1 \ \vee \ p_2]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if } [\![p_1]\!]^A\sigma = \mathtt{tt} \ \text{ or } \ [\![p_2]\!]^A\sigma = \mathtt{tt} \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$[\![\exists \mathtt{x}[p]]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if for some } \ a \in A_s, \ [\![p]\!]^A\sigma\{\mathtt{x}/\ a\ \} = \mathtt{tt} \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

$$[\![\forall \mathtt{x}[p]]\!]^A\sigma \ = \begin{cases} \mathtt{tt} & \text{if for all } a \in A_s, \ [\![p]\!]^A\sigma\{\mathtt{x}/a\} = \mathtt{tt} \\ \mathtt{ff} & \text{otherwise} \end{cases}$$

Note that the definition above can be formulated as,

$$\begin{aligned} [\![\neg p_1]\!]^A\sigma \ &= \ \text{not} \ \ ([\![p_1]\!]^A\sigma) \\ [\![p_1 \ \wedge \ p_2]\!]^A\sigma \ &= \ ([\![p_1]\!]^A\sigma \ \text{ and } \ [\![p_2]\!]^A\sigma) \\ [\![p_1 \ \vee \ p_2]\!]^A\sigma \ &= \ ([\![p_1]\!]^A\sigma \ \text{ or } \ [\![p_2]\!]^A\sigma) \\ [\![t_1 = t_2]\!]^A\sigma \ &= \ ([\![t_1]\!]^A\sigma \underset{\mathsf{2vs}}{=} [\![t_2]\!]^A\sigma) \end{aligned}$$

In the above definition, 'and', 'or', 'not' and '$\underset{\mathsf{2vs}}{=}$' are meta-level logical operators. The meaning of '$\underset{\mathsf{2vs}}{=}$' is given by

$$[\![t_1 \underset{\mathsf{2vs}}{=} t_2]\!]^A\sigma \;=\; \begin{cases} \mathsf{tt} & \text{if } [\![t_1]\!]^A\sigma \downarrow a \text{ and } [\![t_2]\!]^A\sigma \downarrow a \\ \mathsf{ff} & \text{otherwise} \end{cases}$$

Lemma 5.2.1 (Definedness for *TPL*(Σ)).

$$[\![(t = t)]\!]^A\sigma = \begin{cases} \mathsf{tt} & \text{if } [\![t]\!]^A\sigma \downarrow \\ \mathsf{ff} & \text{if } [\![t]\!]^A\sigma\uparrow \end{cases}$$

(Compare the definedness lemma(4.4.2) for ***PPL***(Σ).)

5.3 Functionality and substitution lemmas for assertions

The substitution for assertions are the same as for ***PPL***(Σ) (§4.2).

(Compare §3.3 and §4.3).

Lemma 5.3.1 (Functionality lemma for assertions).

$$\text{If} \quad \sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(p)) \quad \text{then} \quad [\![p]\!]^A\sigma \;=\; [\![p]\!]^A\sigma'.$$

Proof. Suppose that $\sigma \approx \sigma'(\mathsf{rel}\ \boldsymbol{Var}(p))$. We want to show that $[\![p]\!]^A\sigma \;=\; [\![p]\!]^A\sigma'$ by structural induction on p. The interesting case is:

$$p \;\equiv\; (t_1 \;=\; t_2)$$

$$\begin{aligned} LHS &\equiv [\![t_1 = t_2]\!]^A\sigma \\ &= ([\![t_1]\!]^A\sigma \underset{\text{2vs}}{=} [\![t_2]\!]^A\sigma\ ) \quad \text{(by definition)} \\ RHS &\equiv [\![t_1 = t_2]\!]^A\sigma' \\ &= ([\![t_1]\!]^A\sigma' \underset{\text{2vs}}{=} [\![t_2]\!]^A\sigma'\ ) \quad \text{(by definition).} \end{aligned}$$

By the functionality lemma for terms,

$[\![t_1]\!]^A\sigma \simeq [\![t_1]\!]^A\sigma'$ and $[\![t_2]\!]^A\sigma \simeq [\![t_2]\!]^A\sigma'$.

Case 1 : Suppose $[\![t_1]\!]^A\sigma \downarrow a$ and $[\![t_2]\!]^A\sigma \downarrow a$.

Then $[\![t_1]\!]^A\sigma' \downarrow a_1$ and $[\![t_2]\!]^A\sigma' \downarrow a_2$,

and so $LHS = \mathtt{tt} = RHS$.

Case 2 : Suppose $[\![t_1]\!]^A\sigma \downarrow a_1$ and $[\![t_2]\!]^A\sigma \downarrow a_2$ and $a_1 \neq a_2$,

Then $[\![t_1]\!]^A\sigma' \downarrow a_1$ and $[\![t_2]\!]^A\sigma' \downarrow a_2$,

and so $LHS = \mathtt{ff} = RHS$.

Case 3 : Suppose $[\![t_1]\!]^A\sigma \uparrow$ or $[\![t_2]\!]^A\sigma \uparrow$.

Then, correspondingly $[\![t_1]\!]^A\sigma' \uparrow$ or $[\![t_2]\!]^A\sigma' \uparrow$,

and so $LHS = \mathtt{ff} = RHS$.

The other cases ($\neg p_1,\ p_1 \wedge p_2, p_1 \vee p_2, \exists x[p_1],\ \forall x[p_1]$) resemble those in the proof for $\boldsymbol{PPL}(\Sigma)$ in Lemma 4.3.1 (although the underlying logic is, of course, completely different) and we therefore omits the proofs. □

Remark 5.3.2. This also generalize the functionality (or "coincidence") lemma for total algebras (*cf.* Remark 4.3.2).

Corollary 5.3.3. If $\mathtt{x} \notin \boldsymbol{Var}(p)$ then $[\![p]\!]^A\sigma\{\mathtt{x}/a\} \simeq [\![p]\!]^A\sigma$.

Lemma 5.3.4 (Substitution lemma for assertions).

$$[\![t]\!]^A\sigma\downarrow \;\Rightarrow\; ([\![p[x/t]]\!]^A\sigma \;=\; [\![p]\!]^A\sigma\{x/[\![t]\!]^A\sigma\})$$

Proof. Assume $[\![t]\!]^A\sigma\downarrow$. We want to prove

$$([\![p[x/t]]\!]^A\sigma \;=\; [\![p]\!]^A\sigma\{x/[\![t]\!]^A\sigma\}) \;=\; \mathtt{tt}.$$

We prove this by structural induction on p. The interesting case is:

$$\begin{aligned}
p \;&\equiv\; (t_1 = t_2)\\
LHS \;&\equiv\; [\![(t_1 \;=\; t_2)[x/t]]\!]^A\sigma\\
&=\; [\![t_1[x/t] \;=\; t_2[x/t]]\!]^A\sigma \text{ (by definition of substitution)}\\
&=\; [\![t_1[x/t]]\!]^A\sigma \;=\; [\![t_2[x/t]]\!]^A\sigma \text{ (by definition)}\\
RHS \;&\equiv\; [\![t_1 = t_2]\!]^A\sigma\{x/[\![t]\!]^A\sigma\}\\
&=\; [\![t_1]\!]^A\sigma\{x/[\![t]\!]^A\sigma\} \;=\; [\![t_2]\!]^A\sigma\{x/[\![t]\!]^A\sigma\} \text{ (by definition)}
\end{aligned}$$

Case 1*:* Suppose $[\![t_1[x/t]]\!]^A\sigma\downarrow a$ and $[\![t_2[x/t]]\!]^A\sigma\downarrow a$

By the substitution lemma for terms (3.2.7).

$[\![t_1[x/t]]\!]^A\sigma \simeq [\![t_1]\!]^A\sigma\{x/[\![t]\!]^A\sigma\}$ and $[\![t_2[x/t]]\!]^A\sigma \simeq [\![t_2]\!]^A\sigma\{x/[\![t]\!]^A\sigma\}$,

So it follows that $[\![t_1]\!]^A\sigma\{x/[\![t]\!]\sigma\}\downarrow a$ and $[\![t_2]\!]^A\sigma\{x/[\![t]\!]^A\sigma\}\downarrow a$,

and so $LHS \;=\; \mathtt{tt} \;=\; RHS$.

Case 2*:* $[\![t_1[x/t]]\!]^A\sigma\downarrow a_1$ and $[\![t_2[x/t]]\!]^A\sigma\downarrow a_2$ and $a_1 \neq a_2$,

and so $LHS \;=\; \mathtt{ff} \;=\; RHS$.

Case 3*:* $[\![t_1[x/t]]\!]^A\sigma'\uparrow$ or $[\![t_2[x/t]]\!]^A\sigma'\uparrow$,

Then $LHS \;=\; \mathtt{ff}$.

By the substitution lemma for terms, it follows that the corresponding

$[\![t_1]\!]^A \sigma\{x/[\![t]\!]^A\sigma\} \uparrow$ or $[\![t_2]\!]^A \sigma\{x/[\![t]\!]^A\sigma\} \uparrow$

So also $RHS = \mathsf{ff}$.

The other cases $(\neg p_1,\ p_1 \wedge p_2, p_1 \vee p_2, \exists x[p_1], \forall x[p_1])$ resemble those in the proof of Lemma 4.3.4 (although again the underlying logic is completely different) and we therefore omits the proofs. □

Remark 5.3.5. This generalizes the substitution lemma for total algebras (*cf.* Remark 4.3.5).

5.4 Hoare formulae

The definition of Hoare formulae, and their *A-validity*, is exactly the same as for ***PPL/SL***(A) (§4.4), even though the underlying logic is different.

5.5 The proof system ***TPL/SL***(A)

A-validity of inferences are defined exactly as for ***PPL/SL***(A) (*cf.* Definition 4.5.3).

Definition 5.5.1 (Proof system *TPL/SL*(A)). The proof system ***TPL/SL***(A) consists of axioms and inference rules (*cf.* Definition 4.5.4):

(a) *Axioms*:

1. (*Assertions*)

All A-valid formulae $p \mapsto q$

2. (*Assignment*)

$$\{(t\ =\ t) \wedge\ \ p[x/t]\}\quad x := t \quad \{p\}$$

(*b*) *Inference rules*:

1. (*Consequence*)

$$\frac{p \mapsto p_1,\quad \{p_1\}S\{q_1\},\quad q_1 \mapsto q}{\{p\}S\{q\}}$$

2. (*Composition*)

$$\frac{\{p\}S_1\{q\},\quad \{q\}S_2\{r\}}{\{p\}S_1; S_2\{r\}}$$

3. (*Conditional*)

$$\frac{(p \mapsto (b = b)),\quad \{p \wedge (b = \mathsf{true}\ )\}S_1\{q\},\quad \{p \wedge (b = \mathsf{false}\ )\}S_2\{q\}}{\{p\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{q\}}$$

Remark 5.5.2. (Comparasion with ***PPL***(Σ)).

These axioms and inference rules are almost the same as for ***PPL***(Σ) (§4.5) except that

(1) the atomic formulae ($t^s = t^s$) take the place of $\mathsf{def}_s(t)$ in ***PPL/SL***(A).

(2) in the conditional inference rule, the condition ($b = b$) replaces $\mathsf{def}_{\mathsf{bool}}(b)$.

5.6 Soundness of the proof system *TPL/SL*(A)

Lemma 5.6.1 **(A-validity of assignment rule).**

$$\models_A \{(t\ =\ t)\ \wedge\ p[x/t]\}\quad x := t\quad \{p\}$$

Proof. Similar to Lemma 4.6.1. □

Lemma 5.6.2 **(A-validity of consequence rule).** The inference

$$\frac{(p \mapsto p_1),\quad \{p_1\}S\{q_1\},\quad (q_1 \mapsto q)}{\{p\}S\{q\}}$$

is A-valid.

Proof. Similar to Lemma 4.6.2. □

Lemma 5.6.3 **(A-validity of sequential composition rule).** The inference

$$\frac{\{p\}S_1\{q\},\quad \{q\}S_2\{r\}}{\{p\}S_1; S_2\{r\}}$$

is A-valid.

Proof. Similar to Lemma 4.6.3. □

Lemma 5.6.4 **(A-validity of conditional rule).** The inference

$$\frac{p \mapsto ((b = b)),\quad \{p \wedge (b = \mathsf{true}\ )\}S_1\{q\},\quad \{p \wedge (b = \mathsf{false}\ )\}S_2\{q\}}{\{p\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{q\}}$$

is A-valid.

Proof. Similar to Lemma 4.6.4. □

Theorem 5.6.5 (Soundness of $\boldsymbol{TPL/SL}(A)$). For all $f \in \boldsymbol{Form}(\Sigma)$,

$$(\boldsymbol{TPL/SL}(A)) \vdash f \quad \Rightarrow \quad \models f$$

Proof. Since the axioms are valid and the proof rules preserve the validity (Lemma 5.6.1–5.6.4), the result follows by induction on the proof length. □

5.7 The weakest precondition

(Compare §4.7).

Definition 5.7.1 (Weakest preconditions). Given a statement S and predicate q on A, a *weakest precondition* with respect to S and q is an assertion, written $\mathsf{wp}_A(S, q)$, which holds at any $\sigma \in \boldsymbol{State}(A)$ (*i.e.* its value at σ is $\mathtt{tt}$) iff

$$[\![S]\!]^A\sigma \downarrow \text{ and } [\![q]\!]^A([\![S]\!]^A\sigma) \;=\; \mathtt{tt}$$

Note that this definition is identical to the definition (4.7.1) for $\boldsymbol{PPL}(\Sigma)$, although the underlying logic is quite different.

Definition 5.7.2 (A-implication and A-equivalence).

(*i*) $p \underset{A}{\models\!\!\!\Rightarrow} q$ iff for all σ ($[\![p]\!]^A\sigma \;=\; \mathtt{tt} \Rightarrow [\![q]\!]^A\sigma \;=\; \mathtt{tt}$)

(*ii*) $p \underset{A}{\models\!\!\!\dashv} q$ iff $p \underset{A}{\models\!\!\!\Rightarrow} p$ and $q \underset{A}{\models\!\!\!\Rightarrow} p$

i.e. for all σ $[\![p]\!]^A\sigma \;= \mathtt{tt} \quad \Leftrightarrow \quad [\![q]\!]^A\sigma \;=\; \mathtt{tt}$

Definition 5.7.3 (A-equivalence). (a) p and q are A-equivalent iff for all $\sigma \in$ ***State***(A) :

$$[\![p]\!]^A\sigma \;=\; \mathsf{tt} \;\underset{A}{\models\!\!\!\dashv}\; [\![q]\!]^A\sigma \;=\; \mathsf{tt}$$

Note that in ***TPL***(Σ), there are not two concepts of A-implication and A-equivalence ("weak" and "strong") as in ***PPL***(Σ) (compare §4.7).

Again, we have

Lemma 5.7.4. For all $q \in$ ***Assn***,

(a) $\models_A \;\; \{ \; \mathsf{wp}_A(S,q) \; \} \; S \; \{ \; q \; \}$

(b) For each r, if $\models_A \;\; \{ \; r \; \} \; S \; \{ \; q \; \}$, then $\models_A \;\; r \mapsto \; \mathsf{wp}_A(S,q)$

The proof resembles that of Lemma 4.7.5.

Corollary 5.7.5 (Uniqueness of weakest precondition). Any two weakest precondition of S and q on A are A-equivalent (*cf.* Corollary 4.7.6).

We can therefore talk about "the" weakest precondition of S and q on A. We can now show that the preconditions in our proof rules are "best possible".

Lemma 5.7.6. For each $q \in$ ***Assn***,

$$\mathsf{wp}_A(x := t, q) \quad \underset{A}{\models\!\!\!\dashv} \quad (t \;=\; t) \;\;\wedge\;\; q[\mathtt{x}/t]$$

Proof. This follows directly from Definition 5.7.1 and Lemma 5.3.4. □

Lemma 5.7.7. For each $q \in \boldsymbol{Assn}$,

(i) $\mathsf{wp}_A(S_1; S_2, q) \underset{A}{\models\!\!\!\dashv} \mathsf{wp}_A(S_1, (\mathsf{wp}_A(S_2, q))$

(ii) $\mathsf{wp}_A(\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, q) \underset{A}{\models\!\!\!\dashv} \mathsf{if}\ b\ \mathsf{then}\ \mathsf{wp}_A(S_1, q)\ \mathsf{else}\ \mathsf{wp}_A(S_2, q)\ \mathsf{fi}$

Proof. This resembles the proof of Lemma 4.7.9. □

Chapter 6

$\boldsymbol{SLCC}(\Sigma)$ Programs and the Proof System $\boldsymbol{TPL/SLCC}(A)$

In this chapter, we will study the *non-deterministic* programming language $\boldsymbol{SLCC}$ = $\boldsymbol{SLCC}(\Sigma)$ on standard many-sorted partial algebras.

The language $\boldsymbol{SLCC}$ extend $\boldsymbol{SL}$ (Chapter 3) with a new atomic program statement

$$\mathsf{choose}\ \mathsf{z} : b$$

where z:nat and b is a boolean term. Intuitively, this construct selects an arbitrary value for z which makes b true, if there is such a value, and diverges otherwise. This makes $\boldsymbol{SLCC}$ (unlike $\boldsymbol{SL}$) a *nondeterministic* programming language, *i.e.*, from a given initial state more than one execution sequence may be generated.

We give the complete definition of the syntax and semantics of $\boldsymbol{SLCC}$. For the assertions and logic, we choose to work with $\boldsymbol{TPL}(\Sigma)$. (The theory for $\boldsymbol{PPL}(\Sigma)$ is similar, see Remark 6.5.8). Finally, we present the proof system $\boldsymbol{TPL}/\boldsymbol{SLCC}(A)$ and prove its soundness.

Assume Σ is an N-standard signature, and A is an N-standard partial Σ-algebra.

6.1 Syntax of $\boldsymbol{SLCC}(\Sigma)$

Like ***SL***, ***SLCC*** has four syntactic classes: *variables*, *terms*, *statements* and *procedures*. The classes $\boldsymbol{Var}(\Sigma)$ and $\boldsymbol{Term}(\Sigma)$ are exactly as in ***SL*** (§3.1). Next:

(c) $\boldsymbol{AtSt} = \boldsymbol{AtSt}(\Sigma)$ is the class of *atomic statements* $S_{\mathsf{at}}, \ldots,$ defined by

$$S_{\mathsf{at}} \;::=\; \mathsf{skip} \mid \mathtt{x} := t \mid \mathsf{choose}\ \mathsf{z}^{\mathsf{nat}} : b$$

where $\mathtt{x} := t$ is a *concurrent assignment* as before. Note we have a new atomic program statement: '$\mathsf{choose}\ \mathsf{z} : b$' which selects *some* value k such that $b[\mathsf{z}/k]$ is true, if any such k exists (and is undefined otherwise).

In our abstract semantics (§6.2), we will give its meaning as the set of *all possible k's* which make $b[z/k]$ true (hence "countable choice"). Any concrete computation will select a particular k, according to the implementation.

(d) $\boldsymbol{Stmt} = \boldsymbol{Stmt}(\Sigma)$ is the class of statements $S, \ldots,$ defined by:

$$S \;::=\; S_{\mathsf{at}} \mid S_1; S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}.$$

Finally the class $\boldsymbol{Proc}(\Sigma)$ of ***SLCC*** procedures is defined as in ***SL***.

6.2 Semantics of ***SLCC***$(\Sigma\ )$

We interpret ***SLCC*** programs as countably-many-valued state transformations, and ***SLCC*** procedures as defining countably-many-valued functions on A. A many-valued function from A to B can be taken to be a subset of $A \times B$, or (equivalently) a total function from A to $\mathscr{P}_\omega(B)$. We take the latter approach here. We first give some definitions and notations for many-valued functions.

Notation 6.2.1. (a) $\mathscr{P}_\omega(X)$ is the set of all countable subsets of a set X, including the empty set.

(b) $\mathscr{P}^+_\omega(X)$ is the set of all countable *non-empty* subsets of X.

(c) We write $Y^\uparrow$ for $Y \cup \{\uparrow\}$, where '$\uparrow$' denotes divergence.

(d) We write $f : X \rightrightarrows Y$ for $f : X \to \mathscr{P}_\omega(Y)$.

(e) We write $f : X \rightrightarrows^+ Y$ for $f : X \to \mathscr{P}^+_\omega(Y)$.

We will interpret a ***SLCC*** procedure $P : u \to v$ as a countably-many-valued function P^A from A^u to $A^{v\uparrow}$, *i.e.*, as a function

$$P^A :\ A^u \ \to\ \mathscr{P}^+_\omega(A^{v\uparrow})$$

or, in the above notation ([TZ04]):

$$P^A :\ A^u \ \rightrightarrows^+\ A^{v\uparrow}.$$

Definition 6.2.2 (Semantics of statements). We give the detailed semantic definition of $\boldsymbol{SLCC}$ programs.

(a) The meaning of $S_{\mathsf{at}} \in \boldsymbol{AtSt}$ is a function

$$\langle\!| S_{\mathsf{at}} |\!\rangle : \boldsymbol{State}(A) \rightrightarrows^{+} \boldsymbol{State}(A)^{\uparrow}$$

defined by:

$$\langle\!| \mathsf{skip} |\!\rangle^{A} \sigma = \{ \sigma \}$$

$$\langle\!| \mathtt{x} := t |\!\rangle^{A} \sigma \simeq \begin{cases} \{ \sigma\{\mathtt{x}/a\} \} & \text{if } [\![t]\!]^{A}\sigma \downarrow a \\ \{ \uparrow \} & \text{if } [\![t]\!]^{A}\sigma\uparrow \end{cases}$$

$$\langle\!| \mathsf{choose}\ \mathtt{z} : b |\!\rangle^{A} \sigma \simeq \begin{cases} \{ \sigma\{\mathtt{z}/k\} \mid [\![b]\!]^{A}\sigma\{\mathtt{z}/k\} \downarrow \mathtt{tt} \} & \text{if for some } k\ [\![b]\!]^{A}\sigma\{\mathtt{z}/k\} \downarrow \mathtt{tt} \\ \{ \uparrow \} \ \text{otherwise, } \textit{i.e.}\text{, for all } k\ ([\![b]\!]^{A}\sigma\{\mathtt{z}/k\} \downarrow \mathtt{ff} \text{ or } \uparrow) \end{cases}$$

(b) The meaning of $S \in \boldsymbol{Stmt}$ is a function

$$[\![S]\!] : \boldsymbol{State}(A) \rightrightarrows^{+} \boldsymbol{State}(A)^{\uparrow}$$

defined by structural induction on S:

$$[\![S_{\mathsf{at}}]\!]^{A}\sigma \simeq \langle\!| S_{\mathsf{at}} |\!\rangle \sigma$$

$$[\![S_1; S_2]\!]^{A}\sigma \simeq \bigcup \{ [\![S_2]\!]^{A}\sigma' \mid \sigma' \in [\![S_1]\!]^{A}\sigma \} \cup \{ \uparrow \mid \uparrow \in [\![S_1]\!]^{A}\sigma \}$$

$$[\![\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}]\!]^A\sigma \ \simeq \ \begin{cases} [\![S_1]\!]^A\sigma & \text{if } \ [\![b]\!]^A\sigma \downarrow \mathsf{tt} \\ [\![S_2]\!]^A\sigma & \text{if } \ [\![b]\!]^A\sigma \downarrow \mathsf{ff} \\ \{\uparrow\} & \text{otherwise} \end{cases}$$

Lemma 6.2.3. For $S \in \boldsymbol{AtSt}$ and $\sigma \in \boldsymbol{State}(A)$,

$\langle\!|S|\!\rangle\sigma$ is either a singleton set $\{\uparrow\}$ or a non-empty set of proper states σ' $(\neq \uparrow)$.

Proof. This depends on the fact that the semantics of boolean terms are single-valued. □

Definition 6.2.4 (Semantics of procedures). If

$$P \ \equiv \ \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

is a procedure of type $u \to v$, then its meaning in A is a function

$$P^A : \ A^u \ \rightrightarrows^+ \ A^{v\uparrow}$$

defined as follow: for $x \in A^u$,

$$P^A(x) \ = \ \{\,\sigma'(\mathsf{b}) \mid \sigma' \in [\![S]\!]^A\sigma\,\} \ \cup \ \{\uparrow \mid \uparrow \in [\![S]\!]^A\sigma\,\}$$

where σ is any state on A such that $\sigma[\mathsf{a}] = x$.

Remark 6.2.5. A ***SLCC*** procedure $P : u \to v$ is *deterministic* on A if for all $x \in A^u$, $P^A(x)$ is a singleton.

6.3 Assertions

We are going to use ***TPL***(Σ) to construct our assertion language. This is exactly the same as in Chapter 5, with the same 2-valued semantics.

6.4 Hoare formulae

The definition of Hoare formulae is the same as in ***TPL/SL***(A) (§5.4). However, since the notion of A-validity of Hoare triples depends on the semantics of statements which is multivalued in SLCC, the A-validity of Hoare formulae in ***TPL/SLCC***(A) is different from that in ***TPL/SL***(A) or ***PPL/SL***(A).

Definition 6.4.1 (Validity of Hoare formulae). Validity of a formula f w.r.t. A, or A-validity of f, written as

$$\models_A \quad f,$$

is defined as:

Case 1: $f \equiv \ p \mapsto q$

$$\models_A \ f \quad \text{iff for all } \sigma, \text{ if } [\![p]\!]^A\sigma \ = \ \mathtt{tt} \text{ then } [\![q]\!]^A\sigma \ = \ \mathtt{tt}$$

Case 2: $f \equiv \ \{\ p\ \}\ S\ \{\ q\ \}$

$$\models_A \ f \text{ iff for all } \sigma, \text{ if } [\![p]\!]^A\sigma \ = \ \mathtt{tt} \text{ then } (\uparrow \notin [\![S]\!]^A\sigma \text{ and for all } \sigma' \in [\![S]\!]^A\sigma,\ [\![q]\!]^A\sigma' \ = \ \mathtt{tt}).$$

Note again that this leads to *total correctness* (*cf.*Remark 4.4.3).

6.5 The proof system $\boldsymbol{TPL/SLCC}(A)$ and its soundness

A-validity of inferences is defined exactly as for $\boldsymbol{TPL/SL}(A)$ (§5.5) or $\boldsymbol{PPL/SL}(A)$ (§4.5).

Definition 6.5.1 (The proof system $\boldsymbol{TPL/SLCC}(A)$). The proof system $\boldsymbol{TPL/SLCC}(A)$ extends $\boldsymbol{TPL/SL}(A)$ (§5.5) by adding an extra axiom for the 'choose' statement:

$$(a)\ 3.\ \ (\mathit{Choose})\quad \{\ \exists \mathsf{z}\ \ (b\ =\ \mathsf{true})\ \}\ \ \mathsf{choose\ z} : b\ \ \{\ b\ =\ \mathsf{true}\ \}$$

To prove the soundness of $\boldsymbol{TPL/SLCC}(A)$, we must re-prove the A-validity of all axioms and inferences, since now the semantics of Hoare formulae has now changed (§6.4).

Lemma 6.5.2 (Validity of 'choose' rule).

$$\models_A\quad \{\ \exists \mathsf{z}\ (b\ = \mathsf{true})\ \}\ \ \mathsf{choose\ z} : b\ \ \{\ b\ =\ \mathsf{true}\ \}$$

Proof. Assume

$$\models_A\quad \exists \mathsf{z}\ (b\ = \mathsf{true}). \tag{6.1}$$

i.e. assume for all σ,

$$\text{for some } k, \quad [\![b]\!]^A\sigma\{\mathsf{z}/k\} \downarrow \mathsf{tt}, \tag{6.2}$$

We want to show that

$$\uparrow \notin [\![\mathsf{choose}\ \mathsf{z} : b]\!]^A\sigma, \tag{6.3}$$

and

$$\text{for all } \sigma' \in [\![\mathsf{choose}\ \mathsf{z} : b]\!]^A\sigma,\ [\![b]\!]^A\sigma' \downarrow \mathsf{tt}. \tag{6.4}$$

To prove (6.3), suppose (6.3) is false, *i.e.*

$$\uparrow \in [\![\mathsf{choose}\ \mathsf{z} : b]\!]^A\sigma, \tag{6.5}$$

By the semantics of the 'choose' statement (§6.2), we have

$$\text{for all } k \in \mathbb{N}, \quad ([\![b]\!]^A\sigma\{\mathsf{z}/k\} \downarrow \mathsf{ff} \text{ or } [\![b]\!]^A\sigma\{\mathsf{z}/k\}\uparrow)$$

which contradicts (6.2).

To prove (6.4), take an arbitrary $\sigma' \in [\![\mathsf{choose}\ \mathsf{z} : b]\!]^A\sigma$, Again by the semantics of the 'choose' statement, σ' must have the form $\sigma\{\mathsf{z}/k\}$ *i.e.* $\sigma' \equiv \sigma\{\mathsf{z}/k\}$ for some

k such that:

$$[\![b]\!]^A\sigma\{\mathtt{z}/k\} \downarrow \mathtt{tt}$$

thus proving (6.4). $\Box$

Lemma 6.5.3 (Validity of assignment rule).

$$\models_A \{\ (t\ =\ t)\ \wedge\ p[\mathtt{x}/t]\ \}\ \mathtt{x} := t\ \{\ p\ \}$$

Proof. Since if $[\![t]\!]^A\sigma\downarrow$, $[\![x := t]\!]^A\sigma$ is the singleton $\{\sigma\{x/[\![t]\!]^A\sigma\}\}$, the proof is very similar to that for the assignment axiom in ***TPL/SLCC***(A) (Lemma 5.7.1). $\Box$

Lemma 6.5.4 (Validity of consequence rule). The inference

$$\frac{(p \mapsto p_1),\quad \{p_1\}S\{q_1\},\quad (q_1 \mapsto q)}{\{p\}S\{q\}}$$

is A-valid.

Proof. Assume

$$\models_A \quad p \mapsto p_1 \tag{6.6}$$

and

$$\models_A \quad \{p_1\}S\{q_1\} \tag{6.7}$$

and

$$\models_A \quad q_1 \mapsto q. \tag{6.8}$$

We want to show that

$$\models_A \quad \{p\}S\{q\}. \tag{6.9}$$

Take any σ and assume $[\![p]\!]^A\sigma \;=\; \mathtt{tt}$. By (6.6)

$$[\![p_1]\!]^A\sigma \;=\; \mathtt{tt},$$

by (6.7)

$$\uparrow \notin [\![S]\!]^A\sigma \text{ and for all } \sigma_1 \in [\![S]\!]^A\sigma,\;\; [\![q_1]\!]^A(\sigma_1) \;=\; \mathtt{tt},$$

and by (6.8), for all $\sigma_1 \in [\![S]\!]^A\sigma$,

$$[\![q]\!]^A(\sigma_1) \;=\; \mathtt{tt},$$

so that (6.9) holds. $\square$

Lemma 6.5.5 (Validity of composition rule). The inference

$$\frac{\{p\}S_1\{q\},\quad \{q\}S_2\{r\}}{\{p\}S_1;S_2\{r\}}$$

is A-valid.

Proof. Assume

$$\models_A \quad \{p\}S_1\{q\} \tag{6.10}$$

and

$$\models_A \quad \{q\}S_2\{r\}. \tag{6.11}$$

We want to show that

$$\models_A \quad \{p\}S_1;S_2\{r\} \tag{6.12}$$

Take any σ such that

$$[\![p]\!]^A\sigma \;=\; \mathtt{tt}.$$

By (6.10),

$$\uparrow\notin [\![S_1]\!]^A\sigma \text{ and for all } \sigma' \in [\![S_1]\!]^A\sigma', [\![q]\!]^A\sigma' \;=\; \mathtt{tt}.$$

By (6.11), for all $\sigma' \in [\![S_1]\!]^A\sigma$, we have that $\uparrow\, \notin [\![S_2]\!]^A\sigma'$ and

$$\text{for all } \sigma'' \in [\![S_2]\!]^A\sigma',\ [\![r]\!]^A\sigma'' \;=\; \mathtt{tt},$$

hence, by the semantics of composition (§6.2)

$$\uparrow \not\in [\![S_1;S_2]\!]^A\sigma \quad \text{and} \quad \text{for all } \sigma'' \in [\![S_1;S_2]\!]^A\sigma,\ \ [\![r]\!]^A\sigma'' \ = \ \mathtt{tt}$$

i.e. (6.12) holds. □

Lemma 6.5.6 (Validity of conditional rule). The inference

$$\frac{p \ \mapsto (b=b),\ \ \{p \wedge\ (b\ =\ \mathsf{true})\}S_1\{q\},\ \ \{p \wedge\ (b\ =\ \mathsf{false})\}S_2\{q\}}{\{p\}\ \mathsf{if}\ b\ \mathsf{then}\ \ S_1\ \mathsf{else}\ \ S_2\ \mathsf{fi}\ \{q\}}$$

is valid.

Proof. Assume

$$\models_A p \ \ \mapsto (b=b) \tag{6.13}$$

and

$$\models_A\ \{p \wedge\ (b\ =\ \mathsf{true})\}S_1\{q\} \tag{6.14}$$

and

$$\models_A\ \{p \wedge\ (b\ =\ \mathsf{false})\}S_2\{q\}. \tag{6.15}$$

We want to show that

$$\models_A \{p\}\ \mathsf{if}\ b\ \mathsf{then}\ \ S_1\ \mathsf{else}\ \ S_2\ \mathsf{fi}\ \{q\}. \tag{6.16}$$

Take any σ and assume that $[\![p]\!]^A\sigma \;=\; \mathsf{tt}$.

By (6.13) $[\![b]\!]^A\sigma \downarrow (\mathsf{tt} \text{ or } \mathsf{ff})$.

Case 1: $[\![b]\!]^A\sigma \downarrow \mathsf{tt}$. By (6.14),

$$\uparrow \notin [\![S_1]\!]^A\sigma \text{ and for all } \sigma' \in [\![S_1]\!]^A\sigma, [\![q]\!]^A\sigma' \;=\; \mathsf{tt}.$$

Case 2: $[\![b]\!]^A\sigma \downarrow \mathsf{ff}$. By (6.15),

$$\uparrow \notin [\![S_2]\!]^A\sigma \text{ and for all } \sigma' \in [\![S_2]\!]^A\sigma, [\![q]\!]^A\sigma' \;=\; \mathsf{tt}.$$

From the semantics of if-then-else-fi (Definition 6.2.2), (6.16) follows. □

Theorem 6.5.7 (Soundness of *TPL/SLCC*(A)). For all $f \in$ ***Form***,

$$(\boldsymbol{TPL/SLCC}(\mathrm{A})\,) \;\vdash\; f \qquad \Rightarrow \qquad \models_A \; f$$

Proof. Since the axioms are valid and the proof rules preserve the validity (Lemmas 6.5.2 – 6.5.6), the result follows by induction on the proof length. □

Remark 6.5.8 (Comparison with *PPL/SLCC*(A)). The proof system ***PPL/SLCC***(A) for partial logic is very similar to ***TPL/SLCC***(A). The only differences are:

(1) The axiom for 'choose' will be

$$\{\; \exists \mathsf{z}\; b\} \;\; \mathsf{choose}\ \mathsf{z} : b \;\; \{\; b \;\}$$

(2) The conditional rule is slightly different (as in Definition 4.5.4).

Of course, the underlying logic and the semantics are completely different.

6.6 Example

To illustrate the use of the 'choose' rule, we close this section with a simple example of a correctness proof of an ***SLCC*** program. This example solves the problem of approximating reals by rationals [TZ04, Example 1.2.3]. Another, more complex, example to illustrate the use of the 'choose' rule is given in Chapter 9.

Example 6.6.1. Let A be the algebra $\mathcal{R}_p^N$ (Example 2.5.3) augmented by the distance function

$$\mathsf{d} : \mathbb{R}^2 \to \mathbb{R}$$

given by

$$\mathsf{d}(x, y) \;=\; |\; x \;-y\; | \qquad (x, y \in \mathbb{R}).$$

Then A is a metric algebra, *i.e.*, an algebra in which each carrier is a metric space such that all the primitive operations are continuous [TZ04]. Let

$$\mathsf{enum}_Q : \mathbb{N} \to \mathbb{Q}$$

be a standard enumeration of the rationals $\mathbb{Q} \subset \mathbb{R}$. Let

$$\mathsf{negexp} \;: \mathbb{N} \to \mathbb{R}$$

be the function

$$\mathsf{negexp}(n) \;=\; 2^{-n}$$

These are both easily definable in ***While***(A) [ZP93]. Alternatively (for the purpose of this example) we may assume they are included in the signature.

We want to compute a function

$$f \;:\; A \times \mathbb{N} \to \mathbb{R}$$

such that

$$f(a,n) \;=\; \text{“some” } r \in \mathbb{Q} \text{ such that } \mathsf{d}(a,r) \;<\; 2^{-n}.$$

Note that f is many-valued.

Here is an ***SLCC*** procedure for computing f. To make the code more readable, we write $2^{-\mathtt{n}}$ for $\mathsf{negexp}(\mathtt{n})$ and $t_1 < t_2$ for $\mathsf{less}_{\mathsf{real}}(t_1, t_2)$.

We also define the predicate “x is rational”:

$$\mathsf{rat}(x) \;\equiv_{\mathsf{df}}\; \exists\, k\ (\mathsf{enum}_Q(k) \;=\; x)$$

```
proc  CHOOSE
  in a:  real,
  in n:  nat,
  out r:  real,
  aux k:  nat
begin
{ $p$ }
  choose k:  d(a, enum$_Q$ (k))  <  $2^{-n}$;
{ $q_2$ }
  z:= enum$_Q$ (k) ;
{ $q_1$ }
{ $q$ }
end.
```

Our postcondition is

$$q \quad \equiv_{\mathsf{df}} \quad \mathsf{rat}(\mathtt{r}) \ \wedge \ \mathsf{d}(\mathsf{a}, \mathtt{r}) < 2^{-\mathtt{n}},$$

and our precondition is

$$p \quad \equiv_{\mathsf{df}} \quad \exists \mathtt{k} \ \mathsf{d}(\mathsf{a}, \mathsf{enum}_Q(\mathtt{k})) \ < \ 2^{-\mathtt{n}}.$$

which is true in A, by the density of $\mathbb{Q}$ in $\mathbb{R}$.

We give the correctness proof of this procedure in the form of assertions interspersed among the statements, instead of a formal proof tree. However it is very easy to transform this to the proof tree formalism. We define

$$q_1 \quad \equiv_{\mathsf{df}} \quad \exists \mathtt{k} \ ((\mathsf{enum}_Q(\mathtt{k}) \ = \ \mathtt{r}) \ \wedge \ (\mathsf{d}(\mathsf{a}, \mathtt{r}) < 2^{-\mathtt{n}})),$$

Note that

$$q_1 \ \underset{A}{\models\!\!\!\!\Rightarrow}\ q.$$

Define

$$q_2 \ \equiv_{\mathsf{df}}\ \mathsf{d}(\mathsf{a}, \mathsf{enum}_Q(\mathtt{k})) < 2^{-\mathtt{n}}.$$

Note that

$$\mathsf{wp}_A(\mathtt{r} := \mathsf{enum}_Q(\mathtt{k}), q_1)$$

$$\underset{A}{\models\!\!\!\dashv}\ ((\mathsf{enum}_Q(\mathtt{k}) \ = \ \mathsf{enum}_Q(\mathtt{k})) \ \wedge \ \mathsf{d}(\mathsf{a}, \mathbf{enum}_Q(\mathtt{k})) < 2^{-\mathtt{n}})$$

$$\underset{A}{\models\!\!\!\dashv}\ q_2$$

since $\mathsf{enum}_Q(\mathtt{k})$ is a total function, and also

$$(\mathsf{enum}_Q(\mathtt{k}) \ = \ \mathsf{enum}_Q(\mathtt{k})) \quad \underset{A}{\models\!\!\!\dashv} \quad \mathsf{true}.$$

By the 'choose' axiom,

$$p \quad \underset{A}{\models\!\!\!\dashv} \quad \exists \mathtt{k}\ \mathsf{d}(\mathsf{a}, \mathsf{enum}_Q(\mathtt{k})) < 2^{-\mathtt{n}}$$

as required.

Discussion 6.6.2. Why do we need 'choose' for such a simple program? Why can we not simply use the 'least' operation to select the *least* such k, *i.e.*,

$$\mu \mathtt{k} : \mathsf{d}(\mathsf{a}, \mathsf{enum}(\mathtt{k})) < 2^{-\mathtt{n}}?$$

The answer is that this condition is *not testable*, since $\mathsf{less}_{\mathsf{real}}$ is a *partial* function. Why can we not make $\mathsf{less}_{\mathsf{real}}$ a total operation to avoid this problem? The answer is that the total function 'less' would not be continuous, and therefore not computable (see the discussion in §1.4).

Chapter 7

$\boldsymbol{WhileCC}(\Sigma)$ Programs and the Proof System $\boldsymbol{TPL}/\boldsymbol{WhileCC}(A)$

This chapter will extend the programming language $\boldsymbol{SLCC}(\Sigma)$ (Chapter 6) to $\boldsymbol{WhileCC}(\Sigma)$ by adding the 'while' statement. We will study its *algebraic operational semantics,* and then present the proof system $\boldsymbol{TPL}/\boldsymbol{WhileCC}(A)$ and prove its soundness.

Assume (as usual) Σ is an N-standard signature, and A is an N-standard partial Σ-algebra.

7.1 Syntax of $\boldsymbol{WhileCC}(\Sigma)$

The class $\boldsymbol{Var}(\Sigma)$ and $\boldsymbol{Term}(\Sigma)$ are exactly as in $\boldsymbol{SL}$ (§3.1). We extended the class $\boldsymbol{Stmt}(\Sigma)$ by:

(c) $\boldsymbol{AtSt} = \boldsymbol{AtSt}(\Sigma)$ is the class of *atomic statements* $S_{\mathsf{at}}, \ldots,$ defined by :

$$S_{\mathsf{at}} \ ::= \ \mathsf{skip} \mid \mathsf{div} \mid \mathsf{x} := t \mid \mathsf{choose}\ \mathsf{z}^{\mathsf{nat}} : b$$

(d) $\boldsymbol{Stmt} = \boldsymbol{Stmt}(\Sigma)$ is the class of statements $S, \ldots,$ defined by :

$$S \ ::= \ S_{\mathsf{at}} \mid \ S_1; S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \mid \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}$$

Note the introduction of the 'div' (for "local divergence") statement. This is just for technical convenience, in defining the '***Rest***' function (§7.2).

7.2 Semantics of *WhileCC*(Σ)

As with ***SLCC***, we will interpret a ***WhileCC*** statement as a *many-valued* state transformation, and a ***WhileCC*** procedure as a countably-many-valued function on A. This approach follows the semantics of [TZ04] with some minor changes.

Definition 7.2.1 (Semantics of atomic statements). We define the meaning of an atomic statement $S_{\mathsf{at}} \in \boldsymbol{AtSt}$ to be a many-valued function

$$\langle\!| S_{\mathsf{at}} |\!\rangle : \ \boldsymbol{State}(A) \ \rightrightarrows^{+} \ \boldsymbol{State}(A)^{\uparrow}.$$

The semantics of *atomic* statements are the same as for ***SLCC*** (Definition 6.2.2).

Definition 7.2.2 (The '*First*' and '*Rest*' operations). To give the semantics of *non-atomic* statements, we need to define two auxiliary functions: ***First*** and $\boldsymbol{Rest}^{A}$.

For a statement S and a state σ, $\boldsymbol{First}(S,\sigma)$ is an atomic statement which gives the *first* step in the execution of S starting at a state σ, and $\boldsymbol{Rest}^A(S,\sigma)$ is a statement which gives the *rest* of the execution of S starting from σ. These operations

$$\boldsymbol{First} : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightarrow \boldsymbol{AtSt}$$

and

$$\boldsymbol{Rest}^A : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \ \rightarrow \ \boldsymbol{Stmt},$$

are defined as follows:

Case 1 : S is atomic.

$$\boldsymbol{First}(S,\sigma) \ = \ \begin{cases} S & \text{if } \langle\!| S |\!\rangle^A \sigma \neq \{\uparrow\} \\ \mathsf{div} & \text{otherwise} \end{cases}$$

$$\boldsymbol{Rest}^A(S,\sigma) \ = \ \begin{cases} \mathsf{skip} & \text{if } \langle\!| S |\!\rangle^A \sigma \neq \{\uparrow\} \\ \mathsf{div} & \text{otherwise, } i.e., \text{ if } \langle\!| S |\!\rangle \sigma = \{\uparrow\} \end{cases}$$

Case 2 : $S \ \equiv \ S_1; S_2$

$$\boldsymbol{First}(S,\sigma) \ = \ \boldsymbol{First}(S_1,\sigma)$$

$$\boldsymbol{Rest}^A(S,\sigma) \;=\; \begin{cases} S_2 & \\ \qquad \text{if } S_1 \text{ is atomic and } \boldsymbol{Rest}^A(S_1,\sigma) = \mathsf{skip} & \\ \mathsf{div} & \\ \qquad \text{if } S_1 \text{ is atomic and } \boldsymbol{Rest}^A(S_1,\sigma) = \mathsf{div} & \\ \boldsymbol{Rest}^A(S_1,\sigma); S_2 & \\ \qquad \text{otherwise, } i.e., \text{ if } S_1 \text{ is not atomic} & \end{cases}$$

Case 3 : $S \;\equiv\;$ if b then S_1 else S_2 fi.

$$\boldsymbol{First}(S,\sigma) \;=\; \begin{cases} \mathsf{skip} & \text{if } [\![b]\!]^A\sigma\downarrow, \\ \mathsf{div} & \text{otherwise, } i.e., \text{ if } [\![b]\!]^A\sigma\uparrow. \end{cases}$$

$$\boldsymbol{Rest}^A(S,\sigma) \;=\; \begin{cases} S_1 & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{tt}, \\ S_2 & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{ff}, \\ \mathsf{div} & \text{otherwise, } i.e. \text{ if } [\![b]\!]^A\sigma\uparrow. \end{cases}$$

Case 4 : $S \;\equiv\;$ while b do S_0 od.

$$\boldsymbol{First}(S) \;=\; \begin{cases} \mathsf{skip} & \text{if } [\![b]\!]^A\sigma\downarrow, \\ \mathsf{div} & \text{otherwise, } i.e. \text{ if } [\![b]\!]^A\sigma\uparrow. \end{cases}$$

$$\boldsymbol{Rest}^A(S,\sigma) \;=\; \begin{cases} S_0; S & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{tt}, \\ \mathsf{skip} & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{ff}, \\ \mathsf{div} & \text{otherwise, } i.e. \text{ if } [\![b]\!]^A\sigma\uparrow. \end{cases}$$

Definition 7.2.3 (Computation step function). From ***First*** we can define the computation step function

$$\boldsymbol{CompStep}^A : \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightrightarrows^+ \boldsymbol{State}(A)^{\uparrow}$$

as

$$\boldsymbol{CompStep}^A(S,\sigma) \simeq \langle\!|\boldsymbol{First}(S,\sigma)|\!\rangle^A \sigma.$$

Note that $\boldsymbol{CompStep}^A$(S,σ) gives one step in the execution of S from σ.

Definition 7.2.4 (Computation tree). We use a *computation tree* to reflect the execution of a ***WhileCC*** statement S from a state σ. It branches according to all possible outcomes of the one-step computation function $\boldsymbol{CompStep}^A(S,\sigma)$. All the intermediate nodes of this tree are labelled by proper states σ ($\neq \uparrow$). It has two kinds of leaves: a *normal leaf* of this tree is labelled by a proper state σ ($\neq \uparrow$), and a *divergent leaf* is labelled by $\uparrow$. The difference between a normal leaf and divergent leaf is that a normal leaf indicates a *terminating computation*, whereas a divergent leaf indicates *local divergence* from executing :

- an assignment $x := t$ in which t is undefined, or
- a boolean test in which the boolean term is undefined, or
- a 'choose' operation (choose z : b) in which no value of z makes b true, or
- the atomic statement 'div'.

Any actual *concrete* computation of statement S at state σ corresponds to one of the paths through this tree. The possibilities for any such path are:

(a) it is finite, ending in a normal leaf containing the *final proper state* of the computation;

(b) it is finite, ending in a divergent leaf indicating *local divergence*;

(c) it is infinite indicating *global divergence*.

We write $\textbf{\textit{CompTree}}^A(S,\sigma)$ for the *computation tree* of S from the state σ.

Lemma 7.2.5.

$$\textbf{\textit{CompTree}}(S,\sigma) = \{\uparrow\} \quad \text{iff} \quad \textbf{\textit{Rest}}^A(S,\sigma) = \mathsf{div}$$

Proof. By structural induction on S. □

Definition 7.2.6 (Computation tree stage function). We define a *computation tree stage* function

$$\textbf{\textit{CompTreeStage}}^A : \textbf{\textit{Stmt}} \times \textbf{\textit{State}}(A) \times \mathbb{N} \ \rightrightarrows^{+} \ \textbf{\textit{State}}(A)^{\uparrow}$$

where $\textbf{\textit{CompTreeStage}}^A(S,\sigma,n)$ represents the first n stages of $\textbf{\textit{CompTree}}^A(S,\sigma)$.

It is defined by recursion on n:

Basis: $\textbf{\textit{CompTreeStage}}^A(S,\sigma,0) = \{\sigma\}$, *i.e.*, just the root labelled by σ.

Recursion step: $\textbf{\textit{CompTreeStage}}^A(S,\sigma,n+1)$ is formed by:

(i) for S atomic:

for each $\sigma' \in \langle\!|S|\!\rangle^A\sigma$, (where σ' may be a state or $\uparrow$), we attach to the root $\{\sigma\}$ the leaf $\{\sigma'\}$

(ii) for S not atomic, we attach to the root $\{\sigma\}$:

(1) if $\boldsymbol{CompStep}^A(S,\sigma) = \{\uparrow\}$, the leaf $\{\uparrow\}$;

(2) if $\boldsymbol{CompStep}^A(S,\sigma) \neq \{\uparrow\}$,

the subtree $\boldsymbol{CompTreeStage}^A(\boldsymbol{Rest}^A(S,\sigma),\sigma',n)$,

for each $\sigma'(\neq\uparrow)$ of $\boldsymbol{CompStep}^A(S,\sigma)$.

$\boldsymbol{CompTreeStage}^A(S,\sigma,n)$ represents the first n stages of $\boldsymbol{CompTree}^A(S,\sigma)$, which is then defined as the "limit" over n of $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$. Note that the leaves (only) of $\boldsymbol{CompTree}^A(S,\sigma)$ may contain "$\uparrow$", indicating "local divergence".

Lemma 7.2.7. If $\boldsymbol{CompTreeStage}^A(S,\sigma,m)$ has a divergent leaf $\{\uparrow\}$, then

(a) for all $n > m$, $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$ has a divergent leaf $\{\uparrow\}$ at the same position.

(b) $\boldsymbol{CompTree}^A(S,\sigma)$ also has a divergent leaf $\{\uparrow\}$ at the same position.

Proof. (a) By induction on $n \geq m$, with the inductive definition of $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$.

(b) This follows immediately by the definition of $\boldsymbol{CompTree}^A(S,\sigma)$. □

Lemma 7.2.8. (a) If $S \in \boldsymbol{AtSt}$, then $\boldsymbol{CompTreeStage}^A(S,\sigma,n+1)$ is formed by attaching to the root $\{\,\sigma\,\}$, the leaf set $\langle\!|S|\!\rangle^A\sigma$.

(b) If $S \equiv S_1; S_2$, then ***CompTreeStage***$^A(S, \sigma, n+1)$ is formed by attaching to each leaf $\sigma'(\neq \uparrow)$ of ***CompTreeStage***$^A(S_1, \sigma, n+1)$, the subtree ***CompTreeStage***$^A(S_2, \sigma', n+1-m)$, where m is the depth of σ' in ***CompTreeStage***$^A(S_1, \sigma, n+1)$. The leaf $\{\uparrow\}$ of ***CompTreeStage***$^A(S_1, \sigma, n+1)$, remains unchanged.

(c) If $S \equiv \textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}$, then ***CompTreeStage***$^A(S, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$,

Case (1): if $[\![b]\!]^A\sigma \downarrow \mathsf{tt}$,

the subtree ***CompTreeStage***$^A(S_1, \sigma, n)$;

Case (2): if $[\![b]\!]^A\sigma \downarrow \mathsf{ff}$,

the subtree ***CompTreeStage***$^A(S_2, \sigma, n)$;

Case (3): if $[\![b]\!]^A\sigma\uparrow$

the leaf $\{\uparrow\}$.

(d) If $S \equiv \textsf{while } b \textsf{ do } S_0 \textsf{ od}$, then ***CompTreeStage***$^A(S, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$,

Case (1): if $[\![b]\!]^A\sigma \downarrow \mathsf{tt}$,

the subtree ***CompTreeStage***$^A(S_0; S, \sigma, n)$;

Case (2): if $[\![b]\!]^A\sigma \downarrow \mathsf{ff}$,

the leaf $\{\sigma\}$;

Case (3): if $[\![b]\!]^A\sigma\uparrow$,

the leaf $\{\uparrow\}$.

Proof. (a) Directly from Definition 7.2.4. Note that the execution of S from σ can diverge, in which case, we add $\{\uparrow\}$ to the root $\{\sigma\}$.

(b) For the paths does not produce the leaf $\{\uparrow\}$, the proof as in [Wan01, Appendix 2]. For the paths produces the divergent leaf $\{\uparrow\}$, we apply Lemma 7.2.6.

(c) *Case* 1: If $[\![b]\!]^A\sigma\downarrow$ ($\mathtt{tt}$ or $\mathtt{ff}$),

$$\boldsymbol{CompStep}^A(S,\sigma) \;\simeq\; \langle\!|\boldsymbol{First}(S,\sigma)|\!\rangle^A\sigma \;\simeq \langle\!|\mathsf{skip}|\!\rangle\sigma \;\simeq \{\sigma\};$$

Case 2: If $[\![b]\!]^A\sigma\uparrow$,

$$\boldsymbol{CompStep}^A(S,\sigma) \;\simeq\; \langle\!|\boldsymbol{First}(S,\sigma)|\!\rangle^A\sigma \;\simeq \{\uparrow\};$$

The result follows from Definition 7.2.4.

(d) The proof is very similar to (c) above. □

Notation 7.2.9. In this thesis, ***change***(S) denotes the set of variables that appear in S on the left-hand side of the assignment and choose statement. *i.e.*, ***change***(S) is the set of variables that can be modified by S.

The following lemma is needed in the soundness proof of the *Invariance* lemma below (Lemma 7.2.13).

Lemma 7.2.10. For $\mathtt{x} \in \boldsymbol{Var}(S)$, $\sigma \in \boldsymbol{State}(A)$, if $\mathtt{x} \notin \boldsymbol{change}(S)$, then for all $\sigma' \in \boldsymbol{CompTreeStage}^A(S,\sigma,n+1)$ ($\sigma' \neq \uparrow$),

$$\sigma'(\mathtt{x}) \;=\; \sigma(\mathtt{x}).$$

Proof. Assume

$$\mathtt{x} \notin \boldsymbol{change}(S) \tag{1}$$

We must show that for all $\sigma' \in \boldsymbol{CompTreeStage}^A(S, \sigma, n+1)$ $(\sigma' \neq \uparrow)$,

$$\sigma'(\mathtt{x}) \;=\; \sigma(\mathtt{x}). \tag{2}$$

We prove this by structural induction on S.

(a) S is atomic

(i) $S \equiv \mathsf{skip}$

By Lemma 7.2.7, $\boldsymbol{CompTreeStage}^A(\mathsf{skip}, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$, the leaf set $\{\langle\!|\mathsf{skip}|\!\rangle^A\sigma\}$.

Since we only consider $\sigma' \neq \uparrow$, By Lemma 6.2.3 and the semantics of atomic statements,

$$\langle\!|\mathsf{skip}|\!\rangle^A\sigma \;=\; \{\sigma\},$$

so holds in this case.

(ii) $S \equiv \mathsf{y} := t$

By Lemma 7.2.7, $\boldsymbol{CompTreeStage}^A(\mathsf{y} := t, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$ the leaf set $\{\langle\!|\mathsf{y} := t|\!\rangle^A\sigma\}$. Since we only consider $\sigma' \neq \uparrow$, by Lemma 6.2.3 and the semantics of atomic statements,

$$\langle\!|\mathsf{y} := t|\!\rangle^A\sigma \;=\; \{\sigma\{\mathsf{y}/[\![t]\!]^A\sigma\}\}.$$

By (1), $\mathtt{y} \not\equiv \mathtt{x}$, and so by Definition 3.2.2 (Variant of state),

$$\sigma\{\mathtt{y}/[\![t]\!]^A\sigma\}(\mathtt{x}) = \sigma(\mathtt{x}),$$

so (2) holds in this case.

$(iii)\quad S \equiv \mathsf{choose}\ \mathsf{z} : b$

By Lemma 7.2.7, ***CompTreeStage***$^A(\mathsf{choose}\ \mathsf{z} : b, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$ the leaf $\{\langle\!|\mathsf{choose}\ \mathsf{z} : b|\!\rangle^A\sigma\}$. Since we only consider $\sigma' \neq \uparrow$, by Lemma 7.2.2 and the semantics of atomic statements,

$$\langle\!|\mathsf{choose}\ \mathsf{z} : b|\!\rangle^A\sigma \;=\; \{\,\sigma\{\mathtt{z}/k\} \mid [\![b]\!]^A\sigma\{\mathtt{z}/k\} \downarrow \mathtt{tt}\,\},$$

and by (1), $\mathtt{z} \not\equiv \mathtt{x}$, hence for all $\sigma\{\mathtt{z}/k\}$,

$$\sigma\{\mathtt{z}/k\}(\mathtt{x}) = \sigma(\mathtt{x}),$$

so (2) holds in this case.

$(b)\quad S \equiv S_1; S_2$

By Lemma 7.2.7, ***CompTreeStage***$^A(S, \sigma, n+1)$ is formed by attaching to each leaf $\{\sigma'\}(\sigma' \neq \uparrow)$ of ***CompTreeStage***$^A(S_1, \sigma, n+1)$, the subtree ***CompTreeStage***$(S_2, \sigma', n+1-m)$ where m is the depth of σ' in ***CompTreeStage***$^A(S_1, \sigma, n+1)$.
By induction hypothesis, for all σ_1 $(\neq \uparrow)$ in ***CompTreeStage***$^A(S_1, \sigma, n+1)$,

$$\sigma_1(\mathtt{x}) = \sigma(\mathtt{x}).$$

Hence for each leaf σ' of ***CompTreeStage***$^A(S_1, \sigma, n+1)$,

$$\sigma'(\mathtt{x}) = \sigma(\mathtt{x}).$$

Take an arbitrary leaf σ'. By induction hypothesis again, for all σ_2 in ***CompTreeStage***$(S_2, \sigma', n+1-m)$,

$$\sigma_2(\mathtt{x}) = \sigma_2(\mathtt{x}),$$

so again (2) holds.

$(c)\quad S \equiv \textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}$

By Lemma 7.2.7, $\boldsymbol{CompTreeStage}^A(\mathtt{y} := t, \sigma, n+1)$ is formed by attaching to the root $\{\sigma\}$,

Case 1: if $[\![b]\!]^A\sigma \downarrow \mathtt{tt}$, the subtree $\boldsymbol{CompTreeStage}^A(S_1, \sigma, n)$;

Case 2: if $[\![b]\!]^A\sigma \downarrow \mathsf{ff}$, the subtree $\boldsymbol{CompTreeStage}^A(S_1, \sigma, n)$;

Case 3: if $[\![b]\!]^A\sigma\uparrow$, the leaf $\{\uparrow\}$.

By the induction hypothesis, for all $\sigma' \in \boldsymbol{CompTreeStage}^A(S_i, \sigma, n)(i = 1, 2)$, we have $\sigma'(\mathtt{x}) = \sigma(\mathtt{x})$, and adding the root $\{\,\sigma\,\}$ will not change this, hence (2) holds.

(d) $\quad S \equiv \textsf{while}\ \ b\ \textsf{do}\ \ S\ \textsf{od}$

This case resembles case (c) above.

□

Definition 7.2.11 (Semantics of statements). From the semantic computation tree, we can easily define the i/o semantics of statements as

$$[\![S]\!]^A : \boldsymbol{State}(A) \rightrightarrows^{+} \boldsymbol{State}(A)^{\uparrow}.$$

Namely,

$[\![S]\!]^A\sigma$ *is the set of states and/or '↑' at all leaves in* $\boldsymbol{CompTree}^A(S, \sigma)$*, together with '↑' if* $\boldsymbol{CompTree}^A(S, \sigma)$ *has an infinite path.*

Note that, by its definition, $[\![S]\!]^A\sigma$ cannot be empty. It will contain (at least) '↑' if there is at least one computation sequence leading to divergence, *i.e.*, a path of the computation tree which is either infinite or ends in a '↑' leaf.

Theorem 7.2.12 (Semantics of statements).

(a) For S atomic, $[\![S]\!]^A\sigma \;\simeq\; \langle\!|S|\!\rangle^A\sigma$, *i.e.*

$$[\![\mathsf{skip}]\!]^A\sigma \;=\; \{\,\sigma\,\}$$

$$[\![\mathtt{x} := t]\!]^A\sigma \;=\; \begin{cases} \{\,\sigma\{\mathtt{x}/a\}\,\} & \text{if } [\![t]\!]^A\sigma\downarrow a \\ \{\,\uparrow\,\} & \text{if } [\![t]\!]^A\sigma\uparrow \end{cases}$$

$$[\![\mathsf{choose}\ \mathsf{z} : b]\!]^A\sigma \;\simeq\; \begin{cases} \{\,\sigma\{\mathsf{z}/k\} \mid [\![b]\!]^A\sigma\{\mathsf{z}/k\}\downarrow \mathsf{tt}\,\} & \text{if for some } k, [\![b]\!]^A\sigma\{\mathsf{z}/k\}\downarrow \mathsf{tt} \\ \{\,\uparrow\,\} & \text{if for all } k, ([\![b]\!]^A\sigma\{\mathsf{z}/k\}\downarrow \mathsf{ff} \text{ or } \uparrow) \end{cases}$$

(b) $[\![S_1; S_2]\!]^A\sigma \;\simeq\; \bigcup\{\,[\![S_2]\!]^A\sigma' \mid \sigma' \in [\![S_1]\!]^A\sigma\,\} \cup \{\uparrow \mid \uparrow \in [\![S_1]\!]^A\sigma\,\}$

(c) $[\![\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}]\!]^A\sigma \;\simeq\; \begin{cases} [\![S_1]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{tt} \\ [\![S_2]\!]^A\sigma & \text{if } [\![b]\!]^A\sigma\downarrow \mathsf{ff} \\ \{\,\uparrow\,\} & \text{otherwise} \end{cases}$

$$(d)\quad [\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma \ \simeq \begin{cases} [\![S;\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma & \text{if } \ [\![b]\!]^A\sigma \downarrow \mathtt{tt} \\ \{\,\sigma\,\} & \text{if } \ [\![b]\!]^A\sigma \downarrow \mathtt{ff} \\ \{\uparrow\} & \text{otherwise} \end{cases}$$

Proof. (a) This is obvious from Lemma 7.2.7 (a).

(b) By Lemma 7.2.7(b), ***CompTree***$^A(S,\sigma)$ is formed by:

(1) attaching to each leaf $\{\sigma'\}$ $(\neq\ \uparrow)$ of ***CompTree***$^A(S_1,\sigma)$, the subtree ***CompTree***$^A(S_2,\sigma')$, and (2) any leaf $\{\uparrow\}$ of ***CompTree***$^A(S_1,\sigma)$.

Hence the leaves of ***CompTree***$^A(S,\sigma)$ are formed by:

(1) the leaves of ***CompTree***$^A(S_2,\sigma')$ for each σ' in ***CompTree***$^A(S_1,\sigma)$, and (2) any leaf $\{\uparrow\}$ of ***CompTree***$^A(S_1,\sigma)$.

If there is an infinite path in ***CompTree***$^A(S_1,\sigma)$ or in ***CompTree***$^A(S_2,\sigma')$ for any $\sigma' \in$ ***CompTree***$^A(S_1,\sigma)$, then this gives an infinite path in ***CompTree***$^A(S,\sigma)$, which indicates global divergence. Similar comments apply to case (c) and (d) below.

(c) By Lemma 7.2.7(c), ***CompTree***$^A(S,\sigma)$ is formed by attaching to the root $\{\sigma\}$:

Case 1: if $[\![b]\!]^A\sigma \downarrow \mathtt{tt}$, the subtree ***CompTree***$^A(S_1,\sigma)$;

Case 2: if $[\![b]\!]^A\sigma \downarrow \mathtt{ff}$, the subtree ***CompTree***$^A(S_2,\sigma)$;

Case 3: if $[\![b]\!]^A\sigma\uparrow$, the leaf $\{\uparrow\}$.

Hence the leaves of ***CompTreeStage***(S,σ) are formed by:

(1) the leaves of the ***CompTreeStage***(S_i,σ) $(i = 1, 2)$,

(2) the leaf $\{\uparrow\}$, if $[\![b]\!]^A\sigma\uparrow$.

(d) This case resembles case (c).

□

Lemma 7.2.13 (Invariance lemma).

$$\text{if} \quad \mathtt{x} \notin \boldsymbol{change}(S) \quad \text{then} \quad \text{for all } \sigma' \in [\![S]\!]^A\sigma, \sigma(\mathtt{x}) \ = \ \sigma'(\mathtt{x})$$

Proof. Immediately from Lemma 7.2.10 and Theorem 7.2.12. □

Definition 7.2.14 (Semantics of procedures). If

$$P \ \equiv \ \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

is a procedure of type $u \to v$, then its meaning in A is a function

$$P^A : \ A^u \ \rightrightarrows^+ \ A^{v\uparrow}$$

defined as follows [TZ00]. For $x \in A^u$,

$$P^A(x) \ = \ \{\,\sigma'(\mathsf{b}) \mid \sigma' \in [\![S]\!]^A\sigma\,\} \ \cup \ \{\,\uparrow \mid \uparrow \in [\![S]\!]^A\sigma\,\}$$

where σ is any state on A such that $\sigma[\mathsf{a}] = x$.

7.3 Assertions

We will continue to use $\boldsymbol{TPL}(\Sigma)$ for our assertion language as in Chapter 6.

7.4 Hoare formulae

The definition of Hoare formulae, and their A-validity, is exactly the same as in Chapter 6.

7.5 The proof system *TPL/ WhileCC*(A) and its soundness

Definition 7.5.1 (Proof system). The proof system ***TPL/ WhileCC***(A) extends ***TPL/SLCC***(A) (§§5.5, 6.5) by adding

(b)4. *(While)* [Har79, TZ88]

$$\frac{\{\ p(\mathsf{z}+1)\ \}\ S\ \{\ p(\mathsf{z})\ \},\ p(\mathsf{z}+1)\ \mapsto\ (b=\mathsf{true}),\ p(0)\ \mapsto (b\ =\ \mathsf{false})}{\{\ \exists \mathsf{z}\ p(\mathsf{z})\ \}\ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \{\ p(0)\ \ \}}$$

We write 'z+1' instead of '$\mathsf{succ\ z}$' for ease of reading.

(b)5. *(Invariance)*

$$\frac{\{\ p\ \}\ S\ \{\ q\ \}}{\{\ p\ \wedge\ r\ \}\ S\ \{q\ \wedge\ r\ \}}$$

where $\boldsymbol{Var}(r)\ \bigcap\ \boldsymbol{change}(S)\ =\ \emptyset$.

To show the soundness of ***TPL/ WhileCC***(A), we must prove the A-validity of *While* and *Invarance* rules.

Notation 7.5.2.

$$(a) \quad \sigma \models p \quad \Leftrightarrow_{\mathsf{df}} \quad [\![p]\!]^A\sigma = \mathtt{tt}$$

$$(b) \quad \models_A \ p \quad \Leftrightarrow_{\mathsf{df}} \quad \text{for all } \sigma,\ [\![p]\!]^A\sigma = \mathtt{tt}$$

Lemma 7.5.3.

$$(1) \ \ \sigma \models (b = \mathsf{true}) \quad \Leftrightarrow \quad [\![b]\!]^A\sigma \downarrow \mathtt{tt}$$

$$(2) \ \ \sigma \models (b = \mathsf{false}) \quad \Leftrightarrow \quad [\![b]\!]^A\sigma \downarrow \mathtt{ff}$$

Proof. These are obvious. □

Lemma 7.5.4 (Validity of *While* rule).

The inference

$$\frac{\{\ p(\mathsf{z}+1)\ \}\ S\ \{\ p(\mathsf{z})\ \},\ p(\mathsf{z}+1)\ \mapsto\ (b = \mathsf{true}),\ p(0)\ \mapsto (b\ =\ \mathsf{false})}{\{\ \exists \mathsf{z}\ p(\mathsf{z})\ \}\ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \{\ p(0)\ \ \}}$$

is A-valid.

Proof. Assume

$$\models_A \quad \{\ p(\mathsf{z}+1)\ \}\ S\ \{\ p(\mathsf{z})\ \ \} \tag{7.1}$$

and

$$\models_A \quad p(\mathsf{z}+1)\ \mapsto\ (b\ =\ \mathsf{true}) \tag{7.2}$$

and

$$\models_A \quad p(0) \mapsto (b \; = \; \mathsf{false}) \tag{7.3}$$

We must show that

$$\models_A \quad \{\ \exists \mathsf{z}\ p(\mathsf{z})\ \}\ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od} \quad \{\ p(0)\ \} \tag{7.4}$$

So for any state σ, assume

$$\sigma \quad \models \quad \exists \mathsf{z}\ p(\mathsf{z}) \tag{7.5}$$

We must show that

$$\uparrow\ \notin [\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A \sigma \tag{7.6}$$

and

$$\text{for all } \sigma' \in [\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A \sigma, \quad \sigma' \ \models \ p(0) \tag{7.7}$$

From (7.5) follows : for some n,

$$\sigma\{\mathsf{z}/n\} \quad \models \quad p(\mathsf{z}), \tag{7.8}$$

i.e., by the Substitution Lemma,

$$\sigma \quad \models \quad p(n) \tag{7.9}$$

So to prove (7.4), we will show

$$(7.9) \quad \Rightarrow \quad (7.6) \text{ and } (7.7)$$

by induction on n.

Base case : $n = 0$.

We have

$$\sigma \quad \models p(0) \tag{7.10}$$

By (7.3) and Lemma 7.5.3, $[\![b]\!]^A\sigma \downarrow \mathsf{ff}$.

By the semantics of 'while' statements (Theorem 7.2.12 (d)),

$$[\![\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}]\!]^A\sigma \quad = \{\sigma\}.$$

and the results follows from (7.10).

Inductive step:

Assume (i.h.) that for n

$$\sigma \quad \models \quad p(n+1). \tag{7.11}$$

From (7.1),

$$\uparrow \notin [\![S]\!]^A\sigma \ \text{ and } \ \text{for all } \sigma' \in [\![S]\!]^A\sigma, \ \ \sigma' \models \ \ p(n) \tag{7.12}$$

i.e.,

$$\sigma \models \ \ \{p(n+1)\}\ S\ \{p(n)\} \tag{7.13}$$

By (i.h.)

$$\uparrow \notin [\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma' \tag{7.14}$$

and

$$\text{for all } \sigma'' \in [\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma', \sigma'' \models p(n) \tag{7.15}$$

i.e.,

$$\sigma' \models \ \ \{p(n)\}\ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \{p(0)\} \tag{7.16}$$

By 7.13 and 7.16 and the validity of composition lemma,

$$\sigma \models \ \ \{p(n+1)\}\ S; \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \{p(0)\} \tag{7.17}$$

By (7.11), (7.2) and Lemma 7.5.3,

$$[\![b]\!]^A\sigma \downarrow \mathsf{tt}, \tag{7.18}$$

By the semantics of 'while' statements again (Theorem 7.2.12),

$$[\![\mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma \ = \ [\![S; \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}]\!]^A\sigma. \tag{7.19}$$

By (7.17) and (7.19),

$$\sigma \models \ \{\ p(n+1)\ \}\ \ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \ \{\ p(0)\ \}$$

This finished the proof of the inductive step and hence the theorem. □

Lemma 7.5.5 (A-validity of invariance rule). The inference

$$\frac{\{\ p\ \}\ S\ \{\ q\ \}}{\{\ p\ \wedge\ r\ \}\ S\ \{q\ \wedge\ r\ \}}$$

where $\boldsymbol{Var}(r) \bigcap \boldsymbol{change}(S) \ = \ \emptyset$, is A-valid.

Proof. Assume

$$\models_A \quad \{\ p\ \}\ S\ \{\ q\ \} \tag{7.20}$$

and

$$\boldsymbol{Var}(r) \bigcap \boldsymbol{change}(S) \ = \ \emptyset \tag{7.21}$$

We want to show that

$$\models_A \quad \{p \ \wedge \ r\} \ S \ \{q \ \wedge \ r\} \tag{7.22}$$

Take any σ, and assume

$$[\![p \ \wedge \ r]\!]^A \sigma \ = \ \mathtt{tt} \tag{7.23}$$

By the semantics of assertions (§5.2),

$$[\![p]\!]^A \sigma \ = \ \mathtt{tt} \tag{7.24}$$

and

$$[\![r]\!]^A \sigma \ = \ \mathtt{tt}. \tag{7.25}$$

By (7.24) and (7.20), we have

$$\uparrow \notin [\![S]\!]^A \sigma \text{ and for all } \sigma' \ \in \ [\![S]\!]^A \sigma, [\![q]\!]^A \sigma' = \mathtt{tt} \tag{7.26}$$

By (7.21) and the invariance lemma (Lemma 7.2.13), we get

$$\text{for all } \sigma' \ \in \ [\![S]\!]^A \sigma, \sigma \ \simeq \ \sigma' \ (\mathsf{rel\ free}(r)) \tag{7.27}$$

By (7.27) and the functionality lemma for assertions (Lemma 5.4.1),

$$\text{for all } \sigma' \in [\![S]\!]^A\sigma, [\![r]\!]^A\sigma \;=\; [\![r]\!]^A\sigma'. \tag{7.28}$$

From (7.25) and (7.28),

$$\text{for all } \sigma' \;\in [\![S]\!]^A\sigma, [\![r]\!]^A\sigma' = \mathtt{tt}. \tag{7.29}$$

From (7.2.4) and (7.29),

$$[\![q \;\wedge\; r]\!]^A\sigma \;=\; \mathtt{tt},$$

Hence (7.22) holds. □

Note that we proved the preservation rule for *nondeterministic* programs using ***TPL***(Σ). We can similarly formulate and prove a preservation rule using ***PPL***(Σ).

Theorem 7.5.6 (Soundness of *TPL/ WhileCC*(A)). For all p,q $\in$ ***Assn***, $s \in$ ***Stmt***

$$(\textbf{\textit{TPL/ WhileCC}}(\mathrm{A})) \vdash \{p\}S\{q\} \qquad \Rightarrow \qquad \models_A \{p\}S\{q\}$$

Proof. Note that ***TPL/ WhileCC***(A) is the extension of ***TPL/SLCC***(A) formed by adding the *While* and *Invariance* inference rules. By Theorem 6.5.7, together with Lemmas 7.5.4 and 7.5.5, all the axioms and inference rule of ***TPL/ WhileCC***(A) are A-valid. The result follows by induction on the proof length. □

Remark 7.5.7 (The invariance rule). The invariance rule was used by Apt and Olderog [AO91] in their (deterministic) system. According to them, this rule can be derived from other rules of their proof system. We do not know if the same argument applies for our (non-deterministic) system.

Remark 7.5.8 (Comparison with *PPL/SLCC*(A)). Again, the system ***PPL/SLCC***(A) is very similar, in spite of the different underlying logic.

In ***PPL/SLCC***(A), the *While* rule has the form

$$\frac{\{\ p(\mathsf{z}+1)\ \}\ S\ \{\ p(\mathsf{z})\ \},\ p(\mathsf{z}+1)\ \mapsto\ b,\ p(0)\ \mapsto \neg b}{\{\ \exists \mathsf{z}\ p(\mathsf{z})\ \}\ \mathsf{while}\ \ b\ \mathsf{do}\ \ S\ \mathsf{od}\ \{\ p(0)\ \ \}}$$

and the *Invariance* rule has the form

$$\frac{\{\ p\ \}\ S\ \{\ q\ \}}{\{\ p\ \mathsf{cand}\ \ r\ \}\ S\ \{q\ \mathsf{cand}\ r\ \}}$$

where $\boldsymbol{Var}(r)\ \bigcap\ \boldsymbol{change}(S)\ =\ \emptyset$.

Chapter 8

Arrays

We must include the array data structure in our programming language because of its importance (*cf.* our case study in Chapter 9). If we have the right notation for dealing with arrays, then reasoning about programs using arrays can be done simply and effectively.

In this chapter, we will use the logic $\boldsymbol{TPL}(\Sigma)$; the treatment for $\boldsymbol{PPL}(\Sigma)$ is similar with small changes.

8.1 One-dimensional arrays as functions

Traditionally, an array has been considered to be a collection of indexed variables which share a common name and have the same sort s. To refer to a particular element in an array, we specify its name and the corresponding index. In this thesis, we are going to use a different viewpoint: the *functional* view of arrays [Gri83, TZ00] whereby an array `a` is a variable whose value is a function from index values (*i.e.*, naturals) to sort s (as in §2.6). Recall from §2.6 that for each *simple* Σ-sort s, we can have an *array* sort s^* with suitable operations.

We write a for *array variable* (of sort s^*), $\mathtt{i}, \mathtt{j}, \mathtt{k}$ for *index variables*, *i.e.*, variables of sort nat, and (more generally) $e, \ldots$ for *index terms*, *i.e.*, terms of sort nat. We also write, eg, $\boldsymbol{Var}(e_1, e_2)$ for $\boldsymbol{Var}(e_1) \cup \boldsymbol{Var}(e_2)$.

Next we give notations to clarify the relationship between both viewpoints. Recall that $\mathsf{a}[e]$ stands for $\mathsf{Ap}(\mathsf{a}, e)$ (§2.6).

Notation 8.1.1. The "index variable assignment"

$$\mathsf{a}[e] :=\ t, \tag{8.1}$$

is just an alternative notation for

$$\mathsf{a} := \mathsf{Update}(\mathsf{a}, e, t). \tag{8.2}$$

Note that

$$[\![\mathsf{a}[e]]\!]^A\sigma \ = \ \mathsf{Ap}([\![\mathsf{a}]\!]^A\sigma, [\![e]\!]^A\sigma)),$$

which is in A_s.

One advantage of using the functional view is that it is conceptually simple, since, instead of considering a collection of index variables, we now only have one kind of variable: the simple variable. Another advantage follows directly from the first one. Since our notion of states remains unchanged, many results and arguments in the previous chapters carry over. In particular, by using the functional view of arrays, the proof rule for the index variable assignment (8.1) can be derived from the proof rule for the simple variable assignment (8.2) as follows:

Lemma 8.1.2 (A-validity of array assignment rule).

$$\models_A \qquad \{(t \ = \ t) \ \wedge \ p[\mathsf{a}/\mathsf{Update}(\mathsf{a}, e, t)]\} \ \mathsf{a}[e] := t \ \{p\}$$

where $\mathsf{a} : s^*, t : s$.

Proof. Immediate from the A-validity of the assignment rule (Lemma 5.7.1.) and Notation 8.1.1. $\square$

Now we prove lemmas for two important special cases of the array assignment rule, which will be used in our case study.

Lemma 8.1.3. If $\mathsf{a} \notin \boldsymbol{Var}(e, t)$, then

$$\models_A \qquad \{t \ = \ t\} \ \mathsf{a}[e] := t \ \{\mathsf{a}[e] = t\}$$

Proof. By Lemma 8.1.2, we have

$$\models_A \quad \{t = t \ \wedge \ (\mathsf{a}[e] = t)[\mathsf{a}/\mathsf{Update}(\mathsf{a}, e, t)]\} \ \mathsf{a}[e] := t \ \{\mathsf{a}[e] = t\}, \tag{8.3}$$

Note the second conjunct of the precondition can be simplified as:

$$\begin{aligned} & (\mathsf{a}[e] = t)[\mathsf{a}/\mathsf{Update}(\mathsf{a}, e, t)] \\ \underset{A}{\models\!\!\!\dashv} \quad & (\mathsf{Update}(\mathsf{a}, e, t)[e] = t) \quad \text{since } \ \mathsf{a} \notin \boldsymbol{Var}(e, t) \\ \underset{A}{\models\!\!\!\dashv} \quad & (t = t) \end{aligned}$$

Since

$$((t = t) \wedge (t = t)) \ \underset{A}{\models\!\!\!\dashv} \ (t = t),$$

the result follows from (8.3). □

Remark 8.1.4. In $\boldsymbol{PPL}(\Sigma)$, we can formulate a similar lemma:
If $\mathsf{a} \notin \boldsymbol{Var}(e, t)$, then

$$\models_A \qquad \{\mathsf{def}(t)\} \ \mathsf{a}[e] := t \ \{\mathsf{a}[e] = t\}$$

Lemma 8.1.5. If $\mathsf{a} \notin \boldsymbol{Var}(e_1, e_2)$, then

$$\models_A \qquad \{(e_2 = e_2)\} \ \mathsf{a}[e_1] := \mathsf{a}[e_2] \ \{\mathsf{a}[e_1] = \mathsf{a}[e_2]\}. \tag{8.4}$$

Proof. By Lemma 8.1.2, we have

$$\models_A \quad \{\mathsf{a}[e_2] = \mathsf{a}[e_2] \wedge (\mathsf{a}[e_1] = \mathsf{a}[e_2])[\mathsf{a}/\mathsf{Update}(\mathsf{a}, e_1, e_2)]\} \ \mathsf{a}[e_1] := \mathsf{a}[e_2] \ \{\mathsf{a}[e_1] = \mathsf{a}[e_2]\}, \tag{8.5}$$

Note the second conjunct of the precondition can be simplified to:

$$\begin{aligned}
&(\mathsf{a}[e_1] = \mathsf{a}[e_2])[\mathsf{a}/\mathsf{Update}(\mathsf{a}, e_1, e_2)] \\
\underset{A}{\models\!\!\dashv} \quad &(\mathsf{Update}(\mathsf{a}, e_1, e_2)[e_1] = (\mathsf{Update}(\mathsf{a}, e_1, e_2)[e_2]) \quad \text{since} \ \mathsf{a} \notin \boldsymbol{Var}(e, t) \\
\underset{A}{\models\!\!\dashv} \quad &\mathsf{a}[e_2] = \mathsf{a}[e_2] \\
\underset{A}{\models\!\!\dashv} \quad &e_2 = e_2
\end{aligned}$$

(see Remark 8.1.6 (b) below) Since

$$((e_2 = e_2) \wedge (e_2 = e_2)) \ \underset{A}{\models\!\!\dashv} \ (e_2 = e_2),$$

the result follows from (8.5). □

Remark 8.1.6. (a) In $\boldsymbol{PPL}(\Sigma)$, the corresponding lemma is:

If $\mathsf{a} \notin \boldsymbol{Var}(e_1, e_2)$, then

$$\models_A \qquad \{\mathsf{def}(e)\}\mathsf{a}[e_1] := \mathsf{a}[e_2] \ \{\mathsf{a}[e_1] = \mathsf{a}[e_2]\}.$$

(b) Note that since array application is always a total function (see discussion in §1.4), we need only check the index e is defined in the antecedent of (8.4).

8.2 Arrays of arrays

We now extend our notation to cover two-dimensional arrays by interpreting them as (one-dimensional) arrays of arrays.

Notation 8.2.1. Let a be a two dimensional array. We write $\mathsf{a}[e_1, e_2]$ for $\mathsf{a}[e_1][e_2]$ and

$$\mathsf{a}[e_1, e_2] :=\ t^s$$

for

$$\mathsf{a} :=\ \mathsf{Update}(\mathsf{a}, e_1, \mathsf{Update}(\mathsf{a}[e_1], e_2, t))$$

Note that

$$[\![\mathsf{a}[e_1, e_2]]\!]^A\sigma \ = \ \mathsf{Ap}(\mathsf{Ap}([\![\mathsf{a}]\!]^A\sigma, [\![e_1]\!]^A\sigma), [\![e_2]\!]^A\sigma).$$

Now we can similarly prove two lemmas for important special cases of the assignment rule for two dimensional arrays.

Lemma 8.2.2. If $\mathsf{a} \notin \boldsymbol{Var}(e_1, e_2)$, then

$$\models_A \quad \{t = t\} \ \ \mathsf{a}[e_1, e_2] := t \ \ \{\mathsf{a}[e_1, e_2] = t\}$$

Proof. The proof resembles that of Lemma 8.1.3. □

Lemma 8.2.3. If $\mathsf{a} \notin \boldsymbol{Var}(e_1, e_2, e_3, e_4)$, then

$$\models_A \quad \{e_3 = e_3 \wedge e_4 = e_4\} \ \mathsf{a}[e_1, e_2] := \mathsf{a}[e_3, e_4] \ \{\mathsf{a}[e_1, e_2] = \mathsf{a}[e_3, e_4]\}$$

Proof. The proof resembles that of Lemma 8.1.5. □

Remark 8.2.4. (a) We can similarly prove two lemmas for two-dimensional arrays in $\boldsymbol{PPL}(\Sigma)$.

(b) We do not have to introduce "double starred" signatures $(\Sigma^*)^*$ and algebras $(A^*)^*$ to contain two dimensional arrays. The reason is that such an algebra can be effectively coded in A^*, since we can effectively code a finite sequence of starred objects of a given sort as a single starred object of the same sort, by using the Lgth operation. More precisely, a sequence $x_0^*, \ldots, x_{k-1}^*$ of elements of A_s^* (for some sort s) can be coded as a *pair* $(y^*, n^*) \in A_s^* \times \mathbb{N}^*$, where $\mathsf{Lgth}(n^*) = k$, and for $0 \le j < k$, $n^*[j] = \mathsf{Lgth}(x_j^*)$, and $\mathsf{Lgth}(y^*) = n^*[0] + \cdots + n^*[k-1]$, and for $1 \le j \le k$ and $0 \le i < n^*[j]$, $y^*[n^*[0] + \cdots + n^*[j-1] + i] = x_j^*[i]$. (See [TZ00] for details.)

8.3 Programming languages and proof systems with arrays: Notation

When we work over signature Σ^*, and Σ^*-algebra A^* with arrays, then we write $\boldsymbol{SL}^*(\Sigma)$, $\boldsymbol{SLCC}^*(\Sigma)$ and $\boldsymbol{WhileCC}^*(\Sigma)$ for $\boldsymbol{SL}(\Sigma^*)$, $\boldsymbol{SLCC}(\Sigma^*)$ and $\boldsymbol{WhileCC}(\Sigma^*)$ respectively. We also have the corresponding proof systems $\boldsymbol{TPL}/\boldsymbol{SL}^*(A)$ for $\boldsymbol{TPL}/\boldsymbol{SL}(A^*)$ etc.

Chapter 9

Case Study

In this chapter, we will first give a ***WhileCC****** program to implement the "Gaussian Elimination" algorithm [FM67, Neu01] in the algebra $\mathcal{R}_p^{N^*}$ (Example 2.6.2). We will then prove its correctness using the logic $\boldsymbol{TPL}(\Sigma)$. The correctness for $\boldsymbol{PPL}(\Sigma)$ would be fairly similar.

The "Gaussian Elimination" algorithm is used to solve a linear algebraic equation $\mathtt{A} * \mathtt{X} = \mathtt{B}$ by LU decomposition, where finding non-zero pivoting element is the key step in the LU decomposition. In total algebras, we can use *partial pivoting strategy*, which is taking the pivot element to be a non-zero element of largest absolute value in a column. Can we do the same thing in partial algebras? Firstly, we cannot find such a "largest" element since less is partial. Secondly, we cannot even find the first non-zero element as the pivot element since eq is partial. Thirdly, Can we make eq and less a total function to avoid this problem? The answer is that the total function eq and less would not be continuous, and therefore by continuity principle, they are not computable (see the discussion in §1.4). That is why we must use the ‘choose’ statement to select a pivot element and implement the “Gaussian Elimination” algorithm.

For ease of reading and displaying, we indulge in some abuse of notations:

(1) For terms t_1 and t_2 with sort nat and real, we write

$$(t_1 \ = \ t_2) \text{ for } \mathsf{eq}(t_1, t_2) = \mathsf{true}$$

$$(t_1 \ \neq \ t_2) \text{ for } \mathsf{eq}(t_1, t_2) = \mathsf{false}$$

$$(t_1 \ < \ t_2) \text{ for } \mathsf{less}(t_1, t_2) = \mathsf{true}$$

(2) In accordance with common practice, we assume that the domain of index of an array $\mathtt{A}$ is $\{1, \ldots, \ell\}$ rather than $\{0, \ldots, \ell - 1\}$, where $\ell = \mathsf{Lgth}(\mathtt{A})$.

(3) We sometimes write $\mathtt{A}_\mathtt{i}^\mathtt{j}$ for two dimensional array application $\mathtt{A}[\mathtt{i}, \mathtt{j}]$.

(4) We write ‘$\models\!\!\mid$’ for ‘$\underset{A}{\models\!\!\mid}$’ and ‘$\models\!\!\!\Rightarrow$’ for ‘$\underset{A}{\models\!\!\!\Rightarrow}$’, etc.

(5) We define

$\mathtt{A} : \mathtt{n} \equiv_{\mathsf{df}} \mathsf{Lgth}(\mathtt{A}) = \mathtt{n}$

$\mathtt{A} : \mathtt{m} * \mathtt{n} \equiv_{\mathsf{df}} \mathsf{Lgth}(\mathtt{A}) = \mathtt{m} \ \wedge \ (\forall \mathtt{i} \ 1 \leq \ \mathtt{i} \leq \mathtt{m} \Rightarrow \mathsf{Lgth}(\mathtt{A}[\mathtt{m}]) = \mathtt{n})$.

(6) We write $\mathtt{A}^{(\mathtt{m})}$ for the leading $\mathtt{m}$-square submatrices of $\mathtt{A}$, where $1 \ \leq \ \mathtt{m} \ \leq \ \mathtt{n}$.

(7) We write $\det(\mathtt{A})$ for the determinant of matrix A. The matrix is nonsingular is defined as the determinant of the matrix is nonzero (i.e.,$\det(\mathtt{A}) \neq 0$).

We give the corresponding proof in the form of a procedure with assertions interspersed rather than a proof tree (*cf.* Example 6.6).

9.1 Outline

Our algorithm [FM67, Neu01] to solve $\mathtt{A} * \mathtt{X} = \mathtt{B}$ with pivoting is as follows:

With *input*: n, A and B,

output: X,

precondition: *pre* $\equiv_{\mathsf{df}}$ $\mathtt{n} > 0 \ \wedge\ \mathtt{A} : \mathtt{n} * \mathtt{n} \ \wedge\ \mathtt{B} : \mathtt{n} \ \wedge\ (\mathsf{det}(\mathtt{A}) \neq 0)$,

postcondition: *post* $\equiv_{\mathsf{df}}$ $\mathtt{A} * \mathtt{X} = \mathtt{B}$.

Step 1: Calculate a permuted normalized triangular factorization $\mathtt{P} * \mathtt{A} = \mathtt{L} * \mathtt{U}$,

Step 2: Solve $\mathtt{L} * \mathtt{Y} = \mathtt{P} * \mathtt{B}$,

Step 3: Solve $\mathtt{U} * \mathtt{X} = \mathtt{Y}$.

The resulting vector X is the solution of $\mathtt{A} * \mathtt{X} = \mathtt{B}$.

Note that the condition $\mathsf{det}(\mathtt{A}) \neq 0$ is not the weakest precondition for the correct working of this algorithm. However it is a reasonable condition and convenient for our purpose.

According to the algorithm above, our program for “Gaussian Elimination with pivoting ” is:

```
proc Gaussian Elimination
in n:  nat
in A:  real**
in B:  real*
aux LU:  real**
aux Y:  real*
aux piv:  nat*
aux i,j,k,l,m,u,z:  nat
```

```
 out X:   real$^*$
 begin
{ Initialize global pivot index array }
 i:=1;
 while (i≤n) do
  piv[i]:=i;
  i:=i+1;
 od;
{ Decomposition with pivoting; }
 i:=1;
 while (i≤n) do
 choose z : $(\mathtt{i} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0)$;
   if(z> i) then
     piv[i],piv[z]:=piv[z],piv[i];
   fi
   $\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} := \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}}$;
   k:=i+1;
   while (k≤ n) do
     $\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[k]}} := (\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[k]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}}) / \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}$ ;
     $\mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[i]}} := \mathtt{A}^{\mathtt{k}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[r]}}$;
     k:=k+1;
   od;
   i:=i+1;
```

```
od;
{ Forward Elimination }
i:=1;
while (i$\leq$ n) do
  Y[i] := B[piv[i]] $- \sum_{j=1}^{i} \mathtt{LU}^{j}_{\mathtt{piv[i]}} * $ Y[j];
  i:=i+1;
od;
{ Backward Elimination }
i:=n;
while (i> 1 $\wedge$ $\mathtt{LU}^{i}_{\mathtt{piv[i]}} \neq 0$ ) do
  X[i] := (Y[i] $- \sum_{j:=i+1}^{n} \mathtt{LU}^{j}_{\mathtt{piv[i]}} * $ X[j])/$\mathtt{LU}^{i}_{\mathtt{piv[i]}}$;
  i:=i-1;
od;
end.
```

According to the program above, we have divided the whole program into four parts, namely S_0, S_1, S_2 and S_3.

```
proc Gaussian Elimination
begin
$S_1$;  { Initialize global index array piv; }
$S_2$;  { Decomposition with pivoting; }
$S_3$;  { Forward Elimination; }
$S_4$;  { Backward Elimination; }
end.
```

Notes S_2 above uses elimination to find `n*n` lower triangular matrix `L` and lower triangular matrix `U` so that `L*U=P*A`, where `P*A` is the matrix `A` with its rows interchanged. The interchange information is stored in the global array `piv`, and the matrices $\mathtt{L} - \mathtt{I}$ and `U` are stored in `LU`. The global index array `piv` is initialized as `piv[i]=i`. During the elimination, we choose the non-zero element in the column as the pivot element, but the rows (equations) are not actually interchanged. The corresponding elements of `piv` are interchanged instead. We then refer to `A[piv[i],j]` instead of `A[i,j]` and `LU[piv[i],j]` instead of `L[i,j]` and `U[i,j]`. During the process of computation, `A` is keep unchanged, and `LU` is updated. Instead of creating two explicit arrays for `L` and `U`, they are stored in the array `LU`.

Remark 9.1.1. Note that not every non-singular matrix `A` has an `L*U`-factorization. However, It follows from standard linear algebra [Neu01, Theorem 2.3.4] that for any non-singular matrix `A` there exists a unit lower triangular matrix `L`, an upper triangular matrix `U` and a permutation matrix `P` such that `A*P = L*U`. For any such `P`, by [Neu01, Theorem 2.1.8], we can conclude all leading square submatrices $(\mathtt{P}*\mathtt{A})^{(\mathtt{m})}$ are non-singular, and the triangular factorization can be calculated recursively by:

$$\begin{cases} \mathtt{U}[\mathtt{i},\mathtt{k}] := (\mathtt{P}*\mathtt{A})[\mathtt{i},\mathtt{k}] - \sum_{\mathtt{j}=1}^{\mathtt{i}-1}\mathtt{L}[\mathtt{i},\mathtt{j}]*\mathtt{U}[\mathtt{j},\mathtt{k}] & \mathtt{k} \geq \mathtt{i} \\ \mathtt{L}[\mathtt{k},\mathtt{i}] := ((\mathtt{P}*\mathtt{A})[\mathtt{k},\mathtt{i}] - \sum_{\mathtt{j}=1}^{\mathtt{i}-1}\mathtt{L}[\mathtt{k},\mathtt{j}]*\mathtt{U}[\mathtt{j},\mathtt{i}])/\mathtt{U}[\mathtt{i},\mathtt{i}] & \mathtt{k} > \mathtt{i} \end{cases}$$

Note that this computation is possible implies that

for $\mathtt{i} = (1,\dots,\mathtt{n})$, there exists `P` such that $\mathtt{U}[\mathtt{i},\mathtt{i}] \neq 0$.

In other words,

for $\mathtt{i} = (1,\dots,\mathtt{n})$, there exists `P` such that $(\mathtt{P}*\mathtt{A})[\mathtt{i},\mathtt{i}] - \sum_{\mathtt{j}=1}^{\mathtt{i}-1}\mathtt{L}[\mathtt{i},\mathtt{j}]*\mathtt{U}[\mathtt{j},\mathtt{i}] \neq 0$.

It is equivalent to $\exists \mathtt{z}\ (\mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}})$

when we use `piv` in the computation.

Discussion 9.1.2. Consider the 'while' loop containing the 'choose' statement above (in S_2). Define

$$\begin{aligned}
&\mathtt{Pivot}(\mathtt{n}, \mathtt{A}, \mathtt{piv}, \mathtt{LU}, \mathtt{i})\\
&\equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge 1 \leq \mathtt{i} \leq \mathtt{n} + 1\\
&\ \Rightarrow\ (\forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i}\ \wedge\ \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}\\
&\wedge\ 1 \leq \mathtt{l} < \mathtt{i}\ \wedge\ \mathtt{l} < \mathtt{m} \leq \mathtt{n}\ \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}\\
&\wedge\ 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})))
\end{aligned}$$

This means:

piv[*r*] *is the pivot index of* `A` *for row* $r = 1, \ldots, \mathtt{i} - 1$*; and*

`LU` gives the first `i` step in the construction of the upper/lower factorization of `A` with respect to `piv` where $1 \leq \mathtt{i} \leq \mathtt{n}$.

The 'choose' statement has pre- and post-condition

$$\{\ p\ \wedge\ \exists \mathtt{z}\ (b = \mathsf{true})\ \}\ \mathsf{choose}\ \mathtt{z} : b\ \{p\ \wedge\ (b = \mathsf{true})\ \}$$

given by the axiom for 'choose' and the invariance rule, where

$$p \equiv_{\mathsf{df}} q_7$$

(defined on page 162) and

$$b \equiv_{\mathsf{df}} \mathtt{i} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0).$$

Let inv_{out} be the loop invariant of the 'while' loop (defined on page 147).

Note the following:

(1) $inv_{out} \underset{A}{\models\!\!\!\!\Rightarrow} \mathtt{Pivot}(\mathtt{n}, \mathtt{A}, \mathtt{piv}, \mathtt{LU}, \mathtt{i})$,

(2) $\mathsf{det}(\mathtt{A}) \neq 0 \ \wedge \ \mathtt{Pivot}(\mathtt{n}, \mathtt{A}, \mathtt{piv}, \mathtt{LU}, 0) \underset{A}{\models\!\!\!\!\Rightarrow} \exists \mathtt{z}\ (b = \mathsf{true})$.

Statement (2) says that, assuming $\mathsf{det}(\mathtt{A}) \neq 0$, *there exists* a pivot index piv[z] available for step $\mathtt{i}+1$ of the loop which can then be selected by the 'choose' statement. This follows form [Neu01, Theorem 2.3.4].

To prove the above ***WhileCC**** program is correct, we first define the intermediate assertion

$$P_0 \ \equiv_{\mathsf{df}} \ \mathtt{n} > 0 \wedge \exists \mathtt{z}(1 \le \mathtt{l} \le \mathtt{n} \wedge \mathtt{l} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

Next we define the intermediate assertion

$$\begin{aligned} &P_4 \\ &\equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge \forall \mathtt{m}, \mathtt{l}(\mathtt{piv} * \mathtt{L} * \mathtt{piv} * \mathtt{U} = \mathtt{piv} * \mathtt{A} \\ &\wedge \ \mathtt{piv} * \mathtt{L} * \mathtt{Y} = \mathtt{B} \\ &\wedge \ \mathtt{piv} * \mathtt{U} * \mathtt{X} = \mathtt{Y}), \end{aligned}$$

Note that $\models_A \ P_4 \ \mapsto \ post$.

In the subsections that follow, we are going to prove

$$\{\ P_0\ \}\ S_1\ \{\ P_1\ \}$$

$$\{\ P_1\ \}\ S_2\ \{\ P_2\ \}$$

$$\{\ P_2\ \}\ S_3\ \{\ P_3\ \}$$

$$\{\ P_3\ \}\ S_4\ \{\ P_4\ \}$$

The correctness of the program follows from applying the consequence and invariance rule. Note also that $\mathtt{A} : \mathtt{n} * \mathtt{n}, \mathtt{B} : \mathtt{n}$ always holds in the process of computation by invariance rule.

To help explain the algorithm and the correctness proof, we give an example which uses our algorithm to decompose and solve a $3 * 3$ matrix. We want to solve that $\mathtt{A} * \mathtt{X} = \mathtt{B}$, where

$$\mathtt{A} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{bmatrix} \quad \text{and} \quad \mathtt{B} = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}.$$

Each subsection begins with the program statement $S_{\mathtt{i}}$, $(\mathtt{i} = 0, 1, 2, 3)$, followed by the example, and ends with the correctness proof.

9.2 Pivot index array Initialization

Program for pivot index array initialization

$S_1 \equiv$

```
i:=1;
while (i≤n) do
  piv[i]:=i;
  i:=i+1;
od;
```

After the execution of S_1, we expect

$$\texttt{piv} = [1, 2, 3].$$

Correctness proof of pivot index array initialization

Let

$$post_0 \quad \equiv_{\mathsf{df}} \quad \texttt{n} > 0$$

$$post_1 \quad \equiv_{\mathsf{df}} \ \forall \texttt{k} \ (\ 1 \leq \texttt{k} \leq \texttt{n} \Rightarrow \ \texttt{piv}[\texttt{k}] = \texttt{k}),$$

let the postcondition be:

$$Q_1 \quad \equiv_{\mathsf{df}} \quad post_0 \wedge post_1,$$

let the loop invariant be:

$$inv_1 \quad \equiv_{\mathsf{df}} \quad \mathtt{n} > 0 \land 1 \leq \mathtt{i} \leq \mathtt{n} + 1 \land \ \forall \mathtt{k} \ (\ 0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \ \mathtt{piv}[\mathtt{k}] \ = \ \mathtt{k}),$$

and let the loop convergent be:

$$p(\mathtt{m}) \quad \equiv_{\mathsf{df}} \quad \mathtt{n} > 0 \land \mathtt{m} + \mathtt{i} = \mathtt{n} + 1 \land \forall \mathtt{k} \ (\ 0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \ \mathtt{piv}[\mathtt{k}] \ = \ \mathtt{k}).$$

We want to show the loop is correct. *i.e.*

$$\models \quad \{ \ \exists \mathtt{m} \ p(\mathtt{m}) \ \} \ \ S_{11} \ \ \{ \ p(0) \ \}$$

where $S_{11} \equiv$

```
while (i≤n) do
  piv[i]:=i;
  i:=i+1;
od;
```

We prove this in three steps.

Step 1:

$$\models \quad p(0) \ \mapsto \ \neg b \tag{9.1}$$

since

$$P(0) \models\!\!\Rightarrow \mathtt{i} = \mathtt{n} + 1$$

so (9.1) holds.

Step 2:

$$\models \quad p(\mathtt{m}+1) \;\mapsto\; b \tag{9.2}$$

since for all m,

$$\mathtt{m}+1+\mathtt{i}=\mathtt{n}+1 \Rightarrow \; \mathtt{i} \leq \mathtt{n}.$$

so (9.2) holds.

Step 3: We want to prove

$$\models \quad \{\ p(\mathtt{m}+1)\ \}\;\; S_{12} \;\; \{\ p(\mathtt{m})\ \} \tag{9.3}$$

where $S_{12} \equiv$

```
piv[i]:=i;
i:=i+1;
```

$p'(\mathtt{m})$

$\models\!\mid \;\; \mathsf{wp}(\mathtt{i} := \mathtt{i}+1, p(\mathtt{m}))$

$\models\!\mid \mathtt{n}>0 \wedge \mathtt{m}+\mathtt{i}+1=\mathtt{n}+1 \wedge \forall \mathtt{k}\ (0 \leq \mathtt{k} < \mathtt{i}+1 \Rightarrow \mathtt{piv}[\mathtt{k}]=\mathtt{k})$

$\models\!\mid \mathtt{n}>0 \wedge \mathtt{m}+\mathtt{i}+1=\mathtt{n}+1 \wedge \forall \mathtt{k}\ (0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \mathtt{piv}[\mathtt{k}]=\mathtt{k}) \wedge (\mathtt{k}=\mathtt{i}) \Rightarrow \mathtt{piv}[\mathtt{k}]=\mathtt{k})$

$\models\!\mid \mathtt{n}>0 \wedge \mathtt{m}+\mathtt{i}+1=\mathtt{n}+1 \wedge \forall \mathtt{k}\ (0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \mathtt{piv}[\mathtt{k}]=\mathtt{k}) \wedge \mathtt{piv}[\mathtt{i}]=\mathtt{i}).$

According to our function view of arrays in §8, the statement $\mathtt{piv}[\mathtt{i}] := \mathtt{i}$ is the abbreviation of $\mathtt{piv} := \mathsf{Update}(\mathtt{piv}, \mathtt{i}, \mathtt{i})$, hence

$$
\begin{aligned}
& p''(\mathtt{m}) \\
& \models\!\!\dashv\ \mathsf{wp}(\mathtt{piv}[\mathtt{i}] := \mathtt{i}, p'(\mathtt{m})) \\
& \models\!\!\dashv\ \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} + 1 = \mathtt{n} + 1 \wedge \forall \mathtt{k}\ (0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \mathtt{piv}[\mathtt{k}] = \mathtt{k}) \wedge\ \mathsf{Update}(\mathtt{piv}, \mathtt{i}, \mathtt{i})[\mathtt{i}] = \mathtt{i}) \\
& \models\!\!\dashv\ \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} + 1 = \mathtt{n} + 1 \wedge \forall \mathtt{k}\ (0 \leq \mathtt{k} < \mathtt{i} \Rightarrow \mathtt{piv}[\mathtt{k}] = \mathtt{k})
\end{aligned}
$$

Recall

$$p(\mathtt{m}+1) \ \models\!\!\dashv\ \mathtt{n} > 0 \wedge \mathtt{m} + 1 + \mathtt{i} = \mathtt{n} + 1 \wedge \forall \mathtt{k}\ (\ 0 \leq \mathtt{k} < \mathtt{i} \Rightarrow\ \mathtt{piv}[\mathtt{k}]\ =\ \mathtt{k})$$

It is clear that

$$p(\mathtt{m}+1) \quad \models\!\!\Rightarrow \quad p''(\mathtt{m}).$$

So (9.3) holds. By lemma 8.5.4, the loop is correct. *i.e.*,

$$\models \quad \{\exists \mathtt{m}\ p(\mathtt{m})\} \ \ S_{11} \ \ \{p_0\}.$$

Note that in the following sections, we will omit the obvious similar proof of (9.1) and (9.2), and we will concentrate on proving (9.3).

Next, we want to show

$$p(0) \quad \models\!\!\Rightarrow \quad (\neg b \ \wedge \ q) \tag{9.4}$$

since $p(0)$ is actually equivalent to $\neg b \wedge q$, so (9.4) holds.

Finally, since

$$inv_1 \quad \models\!\!\Rightarrow \quad \exists \mathtt{m}\, p(\mathtt{m})$$

and

$$pre_1$$
$$\models\!\!\dashv \mathsf{wp}(\mathtt{i} := 1, inv_1)$$
$$\models\!\!\dashv \mathtt{n} > 0 \wedge 1 \leq 1 \leq \mathtt{n} + 1 \wedge \forall \mathtt{k}\ (\ 0 \leq \mathtt{k} < 1 \Rightarrow \mathtt{piv}[\mathtt{k}] = \mathtt{k})$$
$$\models\!\!\dashv \mathtt{n} > 0 = post_0$$

So the precondition for statement S_0 is just $\mathtt{n} > 0$.

In summary, we have proved

$$\models \quad \{post_0\} \;\; S_1 \;\; \{post_0 \;\wedge\; post_1\}.$$

9.3 Decomposition with Pivoting

Program for Decompose with Pivoting

```
i:=1;
    while (i≤n) do
    choose z:(i≤ z ≤ n ∧ $\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0$);
      if(z> i) then
```

```
        piv[i],piv[z]:=piv[z],piv[i];
      fi
      $\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} := \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}}$;
      k:=i+1;
      while (k$\leq$ n) do
        $\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[k]}} := (\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[k]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}$ ;
        $\mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[i]}} = \mathtt{A}^{\mathtt{k}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[r]}}$;
        k:=k+1;
      od;
      i:=i+1;
    od;
```

Example of Gaussian elimination with pivoting

We begin with

$$\mathtt{A} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{bmatrix} \quad \text{and} \quad \mathtt{piv} = [\ 1\ 2\ 3\ ] \quad \text{and} \quad \mathtt{LU} = \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}$$

Step 1 : $\mathtt{i} = 1$

$$\mathtt{A}[\mathtt{piv}[\mathtt{z}], 1] - \sum_{r=1}^{1-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}[\mathtt{piv}[\mathtt{r}], 1]$$

$$\mathtt{A}[\mathtt{piv}[1], 1] = \mathtt{A}[1, 1] = 0$$

$$\mathtt{A}[\mathtt{piv}[2], 1] = \mathtt{A}[2, 1] = 2$$

$$\mathtt{A}[\mathtt{piv}[3], 1] = \mathtt{A}[3, 1] = 3$$

Considering the second condition of the 'choose' statement, we can choose $\mathtt{z} = 2$or 3. Suppose we choose $\mathtt{z} = 3$, then after index exchange

$$\mathtt{piv} = [\ 3\ 2\ 1.\ ]$$

Computing the pivot element:

$$\mathtt{LU}[\mathtt{piv}[1], 1] := \mathtt{A}[\mathtt{piv}[1], 1] - \sum_{r=1}^{1-1} \mathtt{LU}[\mathtt{piv}[1], r] * \mathtt{LU}[\mathtt{piv}[\mathtt{r}], 1];$$

$$\mathtt{LU}[3, 1] := \mathtt{A}[3, 1] = 3$$

Now we reach the inner 'while' loop statement.

$$\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[k]}} := (\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[k]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}};$$

$$\mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[i]}} = \mathtt{A}^{\mathtt{k}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[r]}};$$

Substep $1 : \mathtt{k} = 2$

$$\mathtt{LU}[\mathtt{piv}[2], 1] := (\mathtt{A}[\mathtt{piv}[2], 1] - \sum_{r=1}^{1-1} \mathtt{LU}[\mathtt{piv}[2], r] * \mathtt{LU}[\mathtt{piv}[\mathtt{r}], 1])/\mathtt{LU}[\mathtt{piv}[1], 1];$$

$$\mathtt{LU}[\mathtt{piv}[2], 1] := \mathtt{A}[\mathtt{piv}[2], 1]/\mathtt{LU}[\mathtt{piv}[1], 1]$$

$$\texttt{LU}[2,1] := \texttt{A}[2,1]/\texttt{LU}[3,1] = 2/3$$

$$\texttt{LU}[\texttt{piv}[1],2] = \texttt{A}[\texttt{piv}[1],2] - \sum_{r=1}^{1-1} \texttt{LU}[\texttt{piv}[1],r] * \texttt{LU}[\texttt{piv}[\texttt{r}],2];$$

$$\texttt{LU}[3,2] = \texttt{A}[3,2] = 3$$

Substep 2 : $\texttt{k} = 3$

$$\texttt{LU}[\texttt{piv}[3],1] := (\texttt{A}[\texttt{piv}[3],1] - \sum_{r=1}^{1-1} \texttt{LU}[\texttt{piv}[3],r] * \texttt{LU}[\texttt{piv}[\texttt{r}],1])/\texttt{LU}[\texttt{piv}[1],1];$$

$$\texttt{LU}[\texttt{piv}[3],1] := \texttt{A}[\texttt{piv}[3],1]/\texttt{LU}[\texttt{piv}[1],1]$$

$$\texttt{LU}[1,1] := \texttt{A}[1,1]/\texttt{LU}[3,1] = 0$$

$$\texttt{LU}[\texttt{piv}[1],3] = \texttt{A}[\texttt{piv}[1],3] - \sum_{r=1}^{1-1} \texttt{LU}[\texttt{piv}[1],r] * \texttt{LU}[\texttt{piv}[\texttt{r}],3];$$

$$\texttt{LU}[3,3] = \texttt{A}[3,3] = 1$$

So after the first iteration,

$$\texttt{A} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{bmatrix} \quad \text{and} \quad \texttt{piv} = [\ 3\ 2\ 1\ ] \quad \text{and} \quad \texttt{LU} = \begin{bmatrix} 0 & \texttt{X} & \texttt{X} \\ 2/3 & \texttt{X} & \texttt{X} \\ 3 & 3 & 1 \end{bmatrix}$$

Step 2 : $\texttt{i} = 2$

$$\texttt{A}[\texttt{piv}[\texttt{z}],2] - \texttt{LU}[\texttt{piv}[\texttt{z}],1] * \texttt{LU}[\texttt{piv}[1],1]$$

$$= \texttt{A}[\texttt{piv}[2],2] - \texttt{LU}[\texttt{piv}[2],1] * \texttt{LU}[\texttt{piv}[1],1]$$

$$= \mathtt{A}[2,2] - \mathtt{LU}[2,1] * \mathtt{LU}[3,1]$$

$$= \mathtt{A}[2,2] - \mathtt{LU}[2,1] * \mathtt{LU}[3,1] = 4 - 2/3 * 3 = 2$$

$$= \mathtt{A}[1,2] - \mathtt{LU}[1,1] * \mathtt{LU}[3,1] = 2$$

Considering the second condition of the 'choose' statement, we can choose $\mathtt{z} = 2$ or 3. Suppose that we choose $\mathtt{z} = 3$ then after index exchange,

$$\mathtt{piv} = [\ 3\ 1\ 2\ ].$$

Computing the pivot element:

$$\mathtt{LU}[\mathtt{piv}[2], 2] := \mathtt{A}[\mathtt{piv}[2], 2] - \sum_{r=1}^{2-1} \mathtt{LU}[\mathtt{piv}[2], r] * \mathtt{LU}[\mathtt{piv}[\mathtt{r}], 2];$$

$$\mathtt{LU}[\mathtt{piv}[2], 2] := \mathtt{A}[\mathtt{piv}[2], 2] - \mathtt{LU}[\mathtt{piv}[2], 1] * \mathtt{LU}[\mathtt{piv}[1], 1];$$

$$\mathtt{LU}[1, 2] := \mathtt{A}[1, 2] - \mathtt{LU}[1, 1] * \mathtt{LU}[3, 1] = 2 - 0 = 2$$

Entering the inner 'while' loop statement.

Substep: $\mathtt{k} = 3$

$$\mathtt{LU}[\mathtt{piv}[3], 2] := (\mathtt{A}[\mathtt{piv}[3], 2] - \sum_{r=1}^{2-1} \mathtt{LU}[\mathtt{piv}[3], r] * \mathtt{LU}[\mathtt{piv}[\mathtt{r}], 2])/\mathtt{LU}[\mathtt{piv}[2], 2]$$

$$\mathtt{LU}[2, 2] := (\mathtt{A}[2, 2] - \mathtt{LU}[[2, 1] * \mathtt{LU}[3, 2])/\mathtt{LU}[1, 2]$$

$$\mathtt{LU}[2, 2] := (4 - 2/3 * 3)/2 = 1$$

$$\texttt{LU}[\texttt{piv}[2],3] = \texttt{A}[\texttt{piv}[2],3] - \sum_{r=1}^{2-1} \texttt{LU}[\texttt{piv}[2],r] * \texttt{LU}[\texttt{piv}[\texttt{r}],3]$$

$$\texttt{LU}[1,3] = \texttt{A}[1,3] - \texttt{LU}[1,1] * \texttt{LU}[3,3] = 4.$$

So after the second iteration, we have

$$\texttt{A} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{bmatrix} \quad \text{and} \quad \texttt{piv} = [\ 3\ 1\ 2\ ] \quad \text{and} \quad \texttt{LU} = \begin{bmatrix} 0 & 2 & 4 \\ 2/3 & 1 & \texttt{X} \\ 3 & 3 & 1 \end{bmatrix}.$$

Step 3 : $\texttt{i} = 3$

$$\texttt{A}[\texttt{piv}[3],3] - \sum_{r=1}^{3-1} \texttt{LU}[\texttt{piv}[3],r] * \texttt{LU}[\texttt{piv}[\texttt{r}],3]$$

$$= \texttt{A}[2,3] - \texttt{LU}[2,1] * \texttt{LU}[3,3] - \texttt{LU}[2,2] * \texttt{LU}[1,3]$$

$$= 2 - 2/3 * 1 - 1 * 4 = -8/3.$$

Considering the second condition of the '`choose`' statement, we can (only) choose $\texttt{z} = 3$, and the index array remains unchanged.

$$\texttt{LU}^{\texttt{i}}_{\texttt{piv[i]}} := \texttt{A}^{\texttt{i}}_{\texttt{piv[i]}} - \sum_{r=1}^{\texttt{i}-1} \texttt{LU}^{r}_{\texttt{piv[i]}} * \texttt{LU}^{\texttt{i}}_{\texttt{piv[r]}}$$

$$\texttt{LU}[2,3] = -8/3.$$

We cannot enter the while loop. So we have

$$\mathtt{A} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 4 & 2 \\ 3 & 3 & 1 \end{bmatrix} \quad \text{and} \quad \mathtt{piv} = [\,3\ 1\ 2\,] \quad \text{and} \quad \mathtt{LU} = \begin{bmatrix} 0 & 2 & 4 \\ 2/3 & 1 & -8/3 \\ 3 & 3 & 1 \end{bmatrix}.$$

This result is correct. Because the values stored in the array LU can be separated into two parts, which we can represent as:

$$\mathtt{L} = \begin{bmatrix} m & 1 & 0 \\ m & m & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

and

$$\mathtt{U} = \begin{bmatrix} 0 & u & u \\ 0 & 0 & u \\ u & 0 & 0 \end{bmatrix}.$$

Then we have

$\mathtt{L} * \mathtt{piv} * \mathtt{U} = \mathtt{A}$

where

$$\mathtt{piv} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Note that if we use L_T and U_T to represent the triangle matrices comes from L and U by rearrange the rows, we will get

$\mathtt{L}_T = \mathtt{piv} * \mathtt{L}$

$\mathtt{U}_T = \mathtt{piv} * \mathtt{U}$

and we can also have:

$\mathtt{L}_T * \mathtt{U}_T = \mathtt{piv} * \mathtt{A}$.

We go back to our example, we can get our

$$\mathtt{L} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{2}{3} & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

and

$$\mathtt{U} = \begin{bmatrix} 0 & 2 & 4 \\ 0 & 0 & -\frac{8}{3} \\ 3 & 3 & 1 \end{bmatrix}.$$

It is easy to check that

$\mathtt{piv} * \mathtt{L} * \mathtt{piv} * \mathtt{U} = \mathtt{piv} * \mathtt{A}$.

Correctness proof of Gaussian elimination with pivoting

Let

$$\begin{aligned}
&post_2 \quad \equiv_{\mathsf{df}} \\
&\forall \mathtt{m}, \mathtt{l}(1 \le \mathtt{m} \le \mathtt{n} \ \wedge\ \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
&\wedge 1 \le \mathtt{l} \le \mathtt{n} \wedge \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
&\wedge 1 \le \mathtt{m} = \mathtt{l} \le \mathtt{n} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})).
\end{aligned}$$

Let the postcondition be:

$$Q_2 \quad \equiv_{\mathsf{df}} \quad post_0 \ \wedge\ post_2,$$

and the outer loop invariant inv_{out} be :

$$\begin{aligned}
&inv_{out} \\
&\equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge 1 \le \mathtt{i} \le \mathtt{n} + 1 \\
&\wedge \forall \mathtt{m}, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \ \wedge\ \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
&\wedge\ 1 \le \mathtt{l} < \mathtt{i} \ \wedge\ \mathtt{l} < \mathtt{m} \le \mathtt{n} \ \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
&\wedge\ 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
&\wedge\ \exists \mathtt{z}(\mathtt{i} \le \mathtt{l} \le \mathtt{n} \wedge 1 \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0)
\end{aligned}$$

and the outer loop convergent $p_1(\mathtt{u})$ be:

$$
\begin{aligned}
& p_1(\mathtt{u}) \\
& \equiv_{\mathsf{df}}\ \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i}\ \wedge\ \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge\ 1 \leq \mathtt{l} < \mathtt{i}\ \wedge\ \mathtt{l} < \mathtt{m} \leq \mathtt{n}\ \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge\ 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \wedge\ \exists \mathtt{z}(\mathtt{i} \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

We want to prove that

$$\models \quad \{p(\mathtt{u}+1)\}\ S_{out}\ \{p(\mathtt{u})\}.$$

q_1

$\mathrel{\models\!\!\!|}\ \mathsf{wp}(\mathtt{i} := \mathtt{i}+1;\ p_1(\mathtt{u}))$

$\mathrel{\models\!\!\!|}\ \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1$

$$\wedge\, \forall \mathtt{m}\ \mathtt{l}(1 \le \mathtt{m} < \mathtt{i}+1 \wedge \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}$$

$$\wedge\, 1 \le \mathtt{l} < \mathtt{i}+1 \wedge \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}$$

$$\wedge\, 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i}+1 \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \ne 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}))$$

$$\wedge\, \exists \mathtt{z}(\mathtt{i}+1 \le \mathtt{l} \le \mathtt{n} \wedge \mathtt{l} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \ne 0).$$

Since the three cases below (without quantifiers) can be expanded into 6 cases:

$$
\begin{aligned}
&q_1\\
&\models\!\mid\ \mathtt{n} > 0 \land \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1\\
&\land \forall \mathtt{m},\, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \land \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}\\
&\land 1 \le \mathtt{m} = \mathtt{i} \land \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}\\
&\land 1 \le \mathtt{l} < \mathtt{i} \land \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}\\
&\land 1 \le \mathtt{l} = \mathtt{i} \land \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}\\
&\land 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})\\
&\land 1 \le \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}))\\
&\land \exists \mathtt{z}(\mathtt{i} + 1 \le \mathtt{l} \le \mathtt{n} \land \mathtt{l} \le \mathtt{z} \le \mathtt{n} \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

After substituting $\mathtt{i}$ into q_1

$$
\begin{aligned}
& q_1 \\
& \models\!\mid \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m},\, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}) \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

To prove the outer loop is correct, we need to set up our inner loop invariant. Note that only the $\mathtt{piv[i]}^{\mathtt{th}}$ line and $\mathtt{i}^{\mathtt{th}}$ column are changed.

Let the inner loop invariant inv_{in} be:

$$
\begin{aligned}
&inv_{in}\\
&\equiv_{\mathsf{df}} \mathtt{n} > 0 \land \mathtt{i} < \mathtt{k} \leq \mathtt{n} + 1 \land \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1\\
&\land \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \land \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}\\
&\land 1 \leq \mathtt{m} = \mathtt{i} \land \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}}\\
&\land 1 \leq \mathtt{l} < \mathtt{i} \land \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}\\
&\land 1 \leq \mathtt{l} = \mathtt{i} \land \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}\\
&\land 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})\\
&\land 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}))\\
&\land \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \land \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0),
\end{aligned}
$$

Let the inner loop convergent $p_2(\mathtt{u})$ be:

$$
\begin{aligned}
& p_2(\mathtt{u}) \\
& \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge v + \mathtt{k} = \mathtt{n} + 1 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l} (1 \le \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \le \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \le \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \ne 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}) \\
& \wedge 1 \le \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \ne 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})) \\
& \wedge \exists \mathtt{z} (\mathtt{i} + 1 \le \mathtt{l} \le \mathtt{n} \wedge \mathtt{l} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \ne 0).
\end{aligned}
$$

We want to show that

$$\models \quad \{p_2(\mathtt{u}+1)\}\ S_{in}\ \{p_2(\mathtt{u})\}.$$

q_2

$\models\!\mid \mathsf{wp}(\mathtt{k} := \mathtt{k}+1, p_2)$

$\models\!\mid \mathtt{n} > 0 \wedge v + \mathtt{k} + 1 = \mathtt{n} + 1 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1$

$\wedge\, \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}$

$\wedge\, 1 \leq \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k}+1 \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}}$

$\wedge\, 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}$

$\wedge\, 1 \leq \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k}+1 \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}$

$\wedge\, 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})$

$\wedge\, 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}))$

$\wedge\, \exists \mathtt{z}(\mathtt{i}+1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$

which can be expanded into 10 cases:

$$
\begin{aligned}
& q_2 \\
& \models\!\!\!\!\dashv\ \mathtt{n} > 0 \wedge v + \mathtt{k} + 1 = \mathtt{n} + 1 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} = \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} = \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}) \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

After substituting for $\mathtt{k}$ in q_2,

$$
\begin{aligned}
& q_2 \\
& \models \mathtt{n} > 0 \wedge v + \mathtt{k} + 1 = \mathtt{n} + 1 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m},\, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \le \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} = \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{k}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \le \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} = \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[k]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \ne 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}) \\
& \wedge 1 \le \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \ne 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \le \mathtt{l} \le \mathtt{n} \wedge \mathtt{l} \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \ne 0).
\end{aligned}
$$

Now we reach the two inner loop assignment statements and the proof proceeds by using Lemma 8.2.3.

$$q_3 \models\!\mid \mathsf{wp}(\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[k]}} := (\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[k]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[k]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}};$$

$$\mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[i]}} := \mathtt{A}^{\mathtt{k}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{k}}_{\mathtt{piv[r]}},\ q2)$$

$$\models\!\mid \mathtt{n} > 0 \wedge v + \mathtt{k} + 1 = \mathtt{n} + 1 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1$$

$$\wedge \forall \mathtt{m},\, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}$$

$$\wedge 1 \le \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}}$$

$$\wedge 1 \le \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}$$

$$\wedge 1 \le \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}$$

$$\wedge 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})$$

$$\wedge 1 \le \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}))$$

$$\wedge \exists \mathtt{z}(\mathtt{i} + 1 \le \mathtt{l} \le \mathtt{n} \wedge 1 \le \mathtt{z} \le \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

It is obvious that

$$p_2(\mathtt{u} + 1) \models\!\Rightarrow q_3.$$

So we have proved the inner loop is correct.

We also simplify q_3 to 8 cases (like q_1).

$$
\begin{aligned}
& q_4 \underset{A}{\models\!\!\!\mid} \mathsf{wp}(\mathtt{k} := \mathtt{i} + 1, inv_{in}) \\
& \underset{A}{\models\!\!\!\mid}\ \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m},\, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{i} \wedge \mathtt{m} < \mathtt{l} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{l} = \mathtt{i} \wedge \mathtt{l} < \mathtt{m} < \mathtt{k} \Rightarrow \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}) \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

We can also simplify q_4 to 6 cases.

$$q_4$$

$$\models\!\!\mid\ \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1$$

$$\wedge\ \forall \mathtt{m},\ \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}$$

$$\wedge\ 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}$$

$$\wedge\ 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})$$

$$\wedge\ 1 \leq \mathtt{m} = \mathtt{l} = \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} = \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}))$$

$$\wedge\ \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

The proof below proceeds by using Lemma 8.2.3.

$$q_5$$

$$\models\!\mid \mathsf{wp}(\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}} := (\mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}}, q_4)$$

$$\models\!\mid \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1$$

$$\wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}$$

$$\wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}$$

$$\wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}))$$

$$\wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[i]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[i]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0$$

$$\wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

Now we reach the ‘if’ statement. Let

$$
\begin{aligned}
& q_6 \\
& \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l} (1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \wedge \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0 \\
& \wedge \exists \mathtt{z} (\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

The following clearly holds.

$$\models \quad \{\ q_6 \wedge \mathtt{z} > \mathtt{i}\ \} \mathtt{piv[z]}, \mathtt{piv[i]} := \mathtt{piv[i]}, \mathtt{piv[z]}\ \{q_5\}$$

and

$$\models \quad \{\ q_6 \wedge (\mathtt{z} = \mathtt{i})\ \}\ \mathsf{skip}\ \{q_5\}.$$

Now we reach the ‘choose’ statement. By the axiom of choose statement (Lemma 6.5.2) and the invariance rule (Lemma 7.5.5), we have

$$\models \quad \{\ p \wedge\ \exists \mathtt{z}\ (b\ =\ \mathsf{true})\}\ \ \mathsf{choose}\ \mathtt{z} :\ b\ \ \{\ p \wedge\ (b\ =\ \mathsf{true})\ \}$$

So let

$$
\begin{aligned}
& q_7 \\
& \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

Then

$$
\begin{aligned}
& q_8 \equiv_{\mathsf{df}} q_7 \wedge \ \exists\ \mathtt{z}\ (b \ = \mathsf{true}) \\
& \models\!\dashv \mathtt{n} > 0 \wedge \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < \mathtt{i} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge 1 \leq \mathtt{l} < \mathtt{i} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge 1 \leq \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \wedge \exists \mathtt{z}(\mathtt{i} + 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0) \\
& \wedge \exists \mathtt{z}(\mathtt{i} \leq \ \mathtt{z} \leq \mathtt{n} \wedge \ \mathtt{A}^{\mathtt{i}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{i}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

This can be simplify as:

$$
\begin{aligned}
& q_8 \\
& \models\!\!| \ \mathtt{n} > 0 \land \mathtt{u} + \mathtt{i} + 1 = \mathtt{n} + 1 \\
& \land \forall \mathtt{m}, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \land \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \land 1 \le \mathtt{l} < \mathtt{i} \land \mathtt{l} < \mathtt{m} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \land 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \land \exists \mathtt{z}(\mathtt{i} \le \mathtt{l} \le \mathtt{n} \land \mathtt{l} \le \mathtt{z} \le \mathtt{n} \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

Recall that:

$$
\begin{aligned}
& p_1(\mathtt{u} + 1) \\
& \models\!\!| \ \mathtt{n} > 0 \land \mathtt{u} + 1 + \mathtt{i} = \mathtt{n} + 1 \\
& \land \forall \mathtt{m}, \mathtt{l}(1 \le \mathtt{m} < \mathtt{i} \ \land \ \mathtt{m} < \mathtt{l} \le \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \land \ 1 \le \mathtt{l} < \mathtt{i} \ \land \ \mathtt{l} < \mathtt{m} \le \mathtt{n} \ \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \land \ 1 \le \mathtt{m} = \mathtt{l} < \mathtt{i} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \land \ \exists \mathtt{z}(\mathtt{i} \le \mathtt{l} \le \mathtt{n} \land \mathtt{l} \le \mathtt{z} \le \mathtt{n} \land \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}
$$

So we clearly have

$$p_1(\mathtt{u}+1) \quad \models\!\!\Rightarrow \quad p_1(\mathtt{u}).$$

Finally,

$$\begin{aligned}
& P_0 \\
& \mathrel{|\!\!=\!\!|}\ \mathsf{wp}(\mathtt{i} := 1; inv_{out}) \\
& \mathrel{|\!\!=\!\!|}\ \mathtt{n} > 0 \wedge 1 \leq 1 \leq \mathtt{n} + 1 \\
& \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} < 1 \ \wedge\ \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \\
& \wedge\ 1 \leq \mathtt{l} < 1 \ \wedge\ \mathtt{l} < \mathtt{m} \leq \mathtt{n} \ \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}} \\
& \wedge\ 1 \leq \mathtt{m} = \mathtt{l} < 1 \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}})) \\
& \wedge\ \exists \mathtt{z}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).
\end{aligned}$$

It can be simplified as:

$$P_0 \mathrel{|\!\!=\!\!|} \mathtt{n} > 0 \wedge \exists \mathtt{z}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

In summary, we have proved:

$$\models \quad \{post_0 \ \wedge\ post_2'\} \ \ S_2 \ \ \{post_0 \ \wedge\ post_2\}.$$

9.4 Forward Elimination

Program for Forward Elimination

The statement section $S_3 \equiv$

```
i:=1;
while (i$\leq$ n) do
    $\mathtt{Y[i]} := \mathtt{B[piv[i]]} - \sum_{\mathtt{j}=1}^{\mathtt{i}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{Y[j]}$;
    i:=i+1;
od.
```

Correctness proof of Forward Elimination

Let

$$post_3 \;\equiv_{\mathsf{df}}\; \forall \mathtt{l}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})$$

and

$$Q_3 \quad \equiv_{\mathsf{df}} \quad post_0 \wedge post_3$$

$$inv_3 \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge 1 \leq \mathtt{i} \leq \mathtt{n} + 1 \wedge \forall \mathtt{l}(1 \leq \mathtt{l} < \mathtt{n} \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})$$

$$p_3(\mathtt{m}) \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} = \mathtt{n} + 1 \wedge \forall \mathtt{l}(1 < \mathtt{l} \leq \mathtt{i} \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]}).$$

We need to prove the loop is correct, *i.e.*

$$\models \quad \{\ \exists \mathtt{m}\ p_3(\mathtt{m})\ \}\ \ S_3\ \ \{\ p_3(0)\ \}$$

where $S_3 \equiv$

```
while (i$\leq$ n) do
   $\mathtt{Y[i]} := \mathtt{B[piv[i]]} - \sum_{\mathtt{j=1}}^{\mathtt{i}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{Y[j]}$;
   i:=i+1;
od.
```

We have to prove three cases. We only give the proof of

$$\models \quad \{\ p_3(\mathtt{m}+1)\ \}\ \ S_{31}\ \ \{\ p_3(\mathtt{m})\ \}$$

where $S_{31} \equiv$

```
   $\mathtt{Y[i]} := \mathtt{B[piv[i]]} - \sum_{\mathtt{j=1}}^{\mathtt{i}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{Y[j]}$;
   i:=i+1.
```

$$p'$$

$$\models\!\!\!\!\dashv\ \mathsf{wp}(\mathtt{i} := \mathtt{i} + 1, p_3(\mathtt{m}))$$

$$\models\!\!\!\!\dashv\ \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} = \mathtt{n} \wedge \forall \mathtt{l}(1 \leq \mathtt{l} < \mathtt{i} + 1 \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j=1}}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})$$

$$\models\!\!\!\!\dashv\ \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} = \mathtt{n} \wedge \forall \mathtt{l}(1 \leq \mathtt{l} < \mathtt{i} \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j=1}}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})$$

$$\wedge\ \mathtt{Y[i]} = \mathtt{B[piv[i]]} - \sum_{\mathtt{j=1}}^{\mathtt{i}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{Y[j]}$$

$$p'' \models\!\!\dashv \mathtt{wp}(\mathtt{Y[i]} := \mathtt{B[piv[i]]} - \sum_{\mathtt{j}=1}^{\mathtt{i}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{Y[j]}, p')$$

$$\models\!\!\dashv \mathtt{n} > 0 \wedge \mathtt{m} + \mathtt{i} = \mathtt{n} \wedge \forall \mathtt{l}(1 \leq \mathtt{l} < \mathtt{i} \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]}).$$

It is clear that

$$p_3(\mathtt{m}+1) \quad \models\!\Rightarrow \quad p''$$

and

$$inv_3 \quad \models\!\Rightarrow \quad \exists \mathtt{m}\; p_3(\mathtt{m}).$$

Finally,

$$pre_3$$

$$\models\!\!\dashv \mathtt{wp}(\mathtt{i} := 1, inv_3)$$

$$\models\!\!\dashv \mathtt{n} > 0 \wedge 0 \leq \mathtt{i} \leq \mathtt{n} \wedge \forall \mathtt{l}(1 \leq \mathtt{l} < 1 \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})$$

$$\models\!\!\dashv \mathtt{n} > 0 \models\!\!\dashv post_0$$

In summary, we have proved that

$$\models \quad \{post_0\} \;\; S_3 \;\; \{post_0 \;\wedge\; post_3\}.$$

9.5 Backward Elimination

Program for Backward Elimination

$S_4 \equiv$

```
i:=n;
while (i> 0 ) do
  $\mathtt{X[i]} := (\mathtt{Y[i]} - \sum_{\mathtt{j=i+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}$;
  i:=i-1;
od.
```

Correctness proof of Backward Elimination

For the statement S_4, let

$$post_4 \equiv_{\mathsf{df}} \quad \forall \mathtt{l}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j=l+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}})$$

and

$$Q_4 \equiv_{\mathsf{df}} \quad post_0 \wedge post_4$$

$$inv_4$$

$$\equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge 0 \leq \mathtt{i} \leq \mathtt{n} \wedge \forall \mathtt{l}(\mathtt{i} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j=l+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}})$$

$$p_4(\mathtt{m}) \equiv_{\mathsf{df}} \mathtt{n} > 0 \wedge \mathtt{m} = \mathtt{i} \wedge \forall \mathtt{l}(\mathtt{i} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j=l+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}).$$

We need to prove the loop is correct, *i.e.*

$$\models \quad \{\ \exists \mathtt{m}\ p_4(\mathtt{m})\ \}\ \ S_{41}\ \ \{\ p_4(0)\ \}$$

where $S_{41} \equiv$

```
while (i> 0 ) do
   X[i] := (Y[i] − $\sum_{j=i+1}^{n} LU^{j}_{piv[i]} * X[j])/LU^{i}_{piv[i]}$;
   i:=i-1;
od.
```

There are three cases. We only give the proof of

$$\models \quad \{\ p_4(\mathtt{m}+1)\ \}\ \ S_{42}\ \ \{\ p_4(\mathtt{m})\ \}$$

where $S_{42} \equiv$

```
   X[i] := (Y[i] − $\sum_{j=i+1}^{n} LU^{j}_{piv[i]} * X[j])/LU^{i}_{piv[i]}$;
   i:=i-1.
```

$$p' \models\!\!\!\dashv \mathsf{wp}(\mathtt{i} := \mathtt{i}-1, p_4(\mathtt{m}))$$

$$\models\!\!\!\dashv \mathtt{n} > 0 \wedge \mathtt{m} = \mathtt{i}-1 \wedge \forall \mathtt{l}(\mathtt{i}-1 < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j}=\mathtt{l}+1}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}})$$

$$\models\!\!\!\dashv \mathtt{n} > 0 \wedge \mathtt{m} = \mathtt{i}-1 \wedge \forall \mathtt{l}(\mathtt{i} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j}=\mathtt{l}+1}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}})$$

$$\wedge\ \mathtt{X[i]} = (\mathtt{Y[i]} - \sum_{\mathtt{j}=\mathtt{i}+1}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}})$$

$$p'' \mathrel{\models\!\!\!|} \mathtt{wp}(\mathtt{X[i]} := (\mathtt{Y[i]} - \sum_{\mathtt{j=i+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[i]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{i}}_{\mathtt{piv[i]}}, p'$$

$$\mathrel{\models\!\!\!|} \mathtt{n} > 0 \wedge \mathtt{m} = \mathtt{i} - 1 \wedge \forall \mathtt{l}(\mathtt{i} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j=l+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}).$$

It is clear that

$$p_4(\mathtt{m}+1) \quad \models\!\!\Rightarrow \quad p''$$

and

$$inv_4 \quad \models\!\!\Rightarrow \quad \exists \mathtt{m}\; p_4(\mathtt{m}).$$

So,

$$pre_4$$

$$\mathrel{\models\!\!\!|} \mathtt{wp}(\mathtt{i} := \mathtt{n}, inv_4)$$

$$\mathrel{\models\!\!\!|} \mathtt{n} > 0 \wedge 0 \leq \mathtt{n} \leq \mathtt{n} \wedge \forall \mathtt{l}(\mathtt{n} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j=l+1}}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}})$$

$$\mathrel{\models\!\!\!|} \mathtt{n} > 0 \mathrel{\models\!\!\!|} q_0.$$

In summary, we have proved

$$\models \quad \{post_0\} \;\; S_4 \;\; \{post_0 \;\wedge\; post_4\}.$$

9.6 Correctness of Gaussian Elimination

$$P_4 \quad \models\!\mid \quad post_0 \ \wedge \ post_1 \ \wedge \ post_2 \ \wedge \ post_3 \ \wedge \ post_4.$$

From (9.5),

$$\models \quad \{post_0\} \ S_4 \ \{post_0 \ \wedge \ post_4\}.$$

By Lemma 7.2.13,

$$\models \quad \{post_0 \wedge post_1 \wedge post_2 \wedge post_3\} \ S_4 \ \{post_0 \wedge post_1 \wedge post_2 \wedge post_3 \wedge post_4\}.$$

Let

$$P_3 \ \equiv_{\mathsf{df}} \ post_0 \ \wedge \ post_1 \ \wedge \ post_2 \ \wedge \ post_3.$$

From (9.4),

$$\models \quad \{post_0\} \ S_3 \ \{post_0 \wedge post_3\}.$$

By Lemma 7.2.13,

$$\models \quad \{post_0 \ \wedge \ post_1 \ \wedge \ post_2\} \ S_4 \ \{post_0 \wedge \ post_1 \ \wedge \ post_2 \ \wedge \ post_3\}.$$

Let

$$P_2 \;\equiv_{\mathsf{df}}\; post_0 \;\wedge\; post_1 \;\wedge\; post_2.$$

From (9.3)

$$\models \quad \{post_0 \wedge post_2'\} \;\; S_3 \;\; \{post_0 \wedge post_2\}.$$

By Lemma 7.2.13,

$$\models \quad \{post_0 \;\wedge\; post_1 \;\wedge\; post_2'\} \;\; S_2 \;\; \{post_0 \wedge\; post_1 \;\wedge\; post_2\}.$$

Let

$$P_1 \;\equiv_{\mathsf{df}}\; post_0 \;\wedge\; post_1 \wedge\; post_2'.$$

From (9.2)

$$\models \quad \{post_0\} \;\; S_3 \;\; \{post_0 \wedge post_1\}.$$

By Lemma 7.2.13,

$$\models \quad \{post_0 \;\wedge\; post_2'\} \;\; S_1 \;\; \{post_0 \wedge\; post_1 \;\wedge\; post_2\}.$$

So the precondition is as desired:

$$P_0 \;\;\models\!\mid\;\; post_0 \;\wedge\; post'_2.$$

To sum up,

$$\begin{aligned}
&P_4\\
&\models\!\mid \mathtt{n} > 0 \wedge \forall \mathtt{m}, \mathtt{l}(1 \leq \mathtt{m} \leq \mathtt{n} \wedge \mathtt{m} < \mathtt{l} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \textstyle\sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}\\
&\wedge 1 \leq \mathtt{l} \leq \mathtt{n} \wedge \mathtt{l} < \mathtt{m} \leq \mathtt{n} \Rightarrow \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}\\
&\wedge 1 \leq \mathtt{m} = \mathtt{l} \leq \mathtt{n} \Rightarrow (\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}} \neq 0 \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[m]}} = \sum_{r=1}^{\mathtt{m}-1} \mathtt{LU}^{r}_{\mathtt{piv[m]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} + \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[m]}}))\\
&\wedge \forall \mathtt{l}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \Rightarrow \mathtt{Y[l]} = \mathtt{B[piv[l]]} - \sum_{\mathtt{j}=1}^{\mathtt{l}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{Y[j]})\\
&\wedge \forall \mathtt{l}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge \Rightarrow \mathtt{X[l]} = (\mathtt{Y[l]} - \sum_{\mathtt{j}=\mathtt{l}+1}^{\mathtt{n}} \mathtt{LU}^{\mathtt{j}}_{\mathtt{piv[l]}} * \mathtt{X[j]})/\mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[l]}}),
\end{aligned}$$

and

$$P_0 \;\;\models\!\mid\;\; \mathtt{n} > 0 \wedge \exists \mathtt{z}(1 \leq \mathtt{l} \leq \mathtt{n} \wedge 1 \leq \mathtt{z} \leq \mathtt{n} \wedge \mathtt{A}^{\mathtt{l}}_{\mathtt{piv[z]}} - \sum_{r=1}^{\mathtt{l}-1} \mathtt{LU}^{r}_{\mathtt{piv[z]}} * \mathtt{LU}^{\mathtt{l}}_{\mathtt{piv[r]}} \neq 0).$$

Again, since

$$\models_A \;pre \;\mapsto\; P_0,$$

and

$$\models_A \; P_4 \;\mapsto\; post,$$

we have proved the program is correct.

Chapter 10

Conclusion and Future work

10.1 Work done

This thesis presented Hoare logic for *total* correctness of *non-deterministic* programs involving computation over *partial* many-sorted algebras over signature Σ.

The programming language ***WhileCC****(Σ) extends the ***While***(Σ) language with "countable choice", *i.e.*, the nondeterministic 'choose' statement and arrays. Our treatment of the semantics of ***WhileCC****(Σ) formally deals with *partial* functions in programming languages.

Two three-tiered logical systems, namely ***TPL***(Σ) and ***PPL***(Σ), were proposed, including an (apparently) original proof rule for the non-deterministic 'choose' construct. Both use the partial (3-valued) logic at the program boolean level, and total (2-valued) logic at the level of correctness formulae (Hoare triples). They differ at the middle level of assertions: ***TPL***(Σ) uses total (2-valued) logic, whereas ***PPL***(Σ) uses partial (3-valued) logic.

Corresponding to these two logics, two proof systems, namely $\boldsymbol{TPL/WhileCC}^*(A)$ and $\boldsymbol{PPL/WhileCC}^*(A)$, were investigated and their *soundness* for total correctness was proved. It is interesting to note how similar these two proof systems are, notwithstanding their quite different logical foundations.

Finally this formal machinery was applied to a correctness proof for a program with pivot selection in the classical Gaussian elimination algorithm, which makes essential use of the non-deterministic 'choose' construct.

10.2 Work needed to be done

A number of issues and problems, which were not solved in this thesis could form a basis for future research.

(1) *Completeness*

A completeness proof for the Hoare systems was not given, or even attempted, for the following reasons:

(a) Such a proof is likely to be quite difficult and should rather form a separate topic for research.

(b) Further, we do not even know if completeness holds for our systems. We do know these system are strong enough to deal with practical, non-trivial problems, such as Gaussian elimination.

(c) Finally, we are not convinced of the value of such a completeness proof. The Hoare systems considered here (and elsewhere) are not really "proof systems" in the usual sense, since the axioms (in particular the *assertion* axiom scheme) are undecidable (*cf.* Remark 4.5.4); hence the role of a completeness proof is not so clear.

(2) *Weakest Precondition*

A more modest, and perhaps more easily attainable, goal would be to find an expression for the *weakest precondition* (or *strongest postcondition*) for the 'choose' construct. This would be interesting in its own right, and might also help with a possible completeness proof. Again however, we do not know if the weakest precondition (or strongest postcondition) is even expressible in our assertion language.

(3) *Proof rule for procedure call*

Owing to a lack of time, we did not give a proof rule for non-deterministic procedure calls. Such a rule has been given in [AO91] for total algebras. We would have to be check the rule is sound for our non-deterministic language ***WhileCC****, which should not be too difficult. Further, it would be very interesting to modify or create such a rule to deal with *partial* algebras. The program in the case study of Chapter 9 (which guided our choice of rules) did not use such procedure calls for the soundness; however, it would have been clearer if it had used such rules; and so it is a pity that because of time constraints, such a proof rule was not incorporated in our Hoare systems.

On the other hand, we feel that we did achieve significant results.

(4) *Independence of invariance rule*

On a more technical level, it would be interesting to settle the problem of the independence of the invariance rule from the other proof rules of our Hoare systems.

It would be worthwhile to settle at least some of the above questions.

Bibliography

[AO91] K.R. Apt and E.R. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, 1991.

[Apt81] K.R. Apt. Ten years of Hoare's logic, a survey - part I. *ACM Transactions on Programming Language and Systems*, 3:431–483, 1981.

[Apt84] K.R. Apt. Ten years of Hoare's logic, a survey - part II: Nondeterminisim. *Theoretical Computer Science*, 28:83–109, 1984.

[BCJ84] H. Barringer, J.H. Cheng, and C.B. Jones. Logic covering undefinedness in program proofs. *Acta Informatica*, 21:251–269, 1984.

[Coo78] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978.

[dB80] Jaco de Bakker. *Mathematical Theory of Program Correctness.* Prentice-Hall International, 1980.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[Far90] W.F. Farmer. A partial functions version of church's simple theory of types. *J. Symbolic Logic*, 55:1269–1291, Sep. 1990.

[Flo67] R. Floyd. Assigning meanings to programs. In *Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Sociaty, 1967. From Proceedings of Symposium on Applied Mathematics 19.

[FM67] G. Forsythe and C.B. MOLER. *Computer Solution of Linear Algebraic Systems.* Prentice-Hall, 1967.

[Gri83] D. Gries. *The Science of Programming.* Springer-Verlag, 1983.

[Gri97] D. Gries. Eliminating the chaff again. In M.Broy and B.Schieder, editors, *Mathematical Methods in Program Development*, volume 158 of *NATO ASI Series F: Computer and System Sciences*, pages 1–7. Springer, Berlin, 1997.

[Har79] D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science.* Springer-Verlag, New York, 1979.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communication of the ACM*, 12:576–580, 583, 1969.

[Hoo87] A. Hoogewijs. Partial-predicate logic in computer science. *Acta Informatica*, 24:381–393, 1987.

[Jon86] C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International, Englewood Cliffs, New Jersey, 1986.

[Jon87] C.B. Jones. *VDM Proof Obligation and their Justification*, volume 252, pages 260–286. Lecture Notes in Computer Science, 1987.

[KK94] M. Kerber and M. Kohlhase. A mechanization of strong kleene logic for partial functions. In A. Bundy, editor, *Lecture Notes in Computer Science*, volume 184, pages 371–385. Springer-Verlag, 1994.

[Kle52] S.C. Kleene. *Introduction to Metamathematics.* North-Holland, Amsterdam, 1952.

[Luk70] J. Lukasiewicz. *Selected Works.* North-Holland, Amsterdam, 1970. Translate from J. Lukasiewicz 1920's work.

[McC63] J. McCarthy. A base for a methematical theory of computation. In *Computer Programming and Formal Systems.* North-Holland, 1963. An earlier version was published in 1961 in the Proceedings of the Western Joint Computer Conference.

[Neu01] A. Neumaier. *Introduction to Numerical Analysis.* Cambridge University Press, 2001.

[Owe93] O. Owe. Partial logics reconsidered: A conservative approach. *Formal Aspects of Computing*, 5:208–223, 1993.

[Par93] D.L. Parnas. Predicate logic for software engineering. *IEEE Transactions on Software Engineering*, 19:856–862, 1993.

[SA91] V. Sperschneider and G. Antoniou. *Logic: A Foundation for Computer Science.* Addison-Wesley, 1991.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific journal of Mathematics*, 5:285–309, 1955.

[Tur49] A.M. Turing. Checking a large routine. In *High Speed Automatic Calculating Machines*, pages 67–69, Cambridge, 1949. Cambridge University Mathematical Laboratory.

[TZ88] J.V. Tucker and J.I. Zucker. *Program Correctness over Abstract Data Types, with Error-State Semantics.* North-Holland, 1988.

[TZ00] J.V. Tucker and J.I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 317–523. Oxford University Press, 2000.

[TZ04] J.V. Tucker and J.I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic*, 2004.

[Wan01] Y. Wang. Semantics of non-deterministic programs and the universal function theorem over abstract algebras. Master's thesis, Dept. of Computing and Software, McMaster University, 2001.

[ZP93] J.I. Zucker and L. Pretorius. Introduction to computability theory. *South African Computer Journal*, 9:3–30, 1993.