Universidade Federal de Pernambuco
Centro de Informática

MSc. in Computer Science

ArcAngel: a Tactic Language For Refinement and its Tool Support
by
Marcel Vinícius Medeiros Oliveira

MSc. Thesis

Recife, 4th December, 2002

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Marcel Vinícius Medeiros Oliveira

ArcAngel: a Tactic Language For Refinement and its Tool Support

This work was presented to the Msc programme in Computer Science, Centro de Informática, Universidade Federal de Pernambuco as a requirement for the award of the degree.

Supervisor: Profa. Ana Lúcia Caneca Cavalcanti

Acknowledgements

Abstract

Morgan's refinement calculus is a successful technique to develop and implement software in a precise, complete, and consistent way. From a formal specification we produce a program which correctly implements the specification by repeatedly applying transformation rules, which are called refinement laws. Using the refinement calculus, however, can be a hard task, as program developments may prove to be long and repetitive.

Frequently used strategies of development are reflected in sequences of law applications that are over and over applied in different developments or even in different points of a single development. A lot is to be gained from identifying these tactics of development, documenting them, and using them in program developments as a single transformation rule.

In this work we present ArcAngel, a language for the definition of refinement tactics based on Angel, and formalize its semantics. Angel is a general-purpose tactic language that assumes only that rules transform proof goals. The semantics of ArcAngel is similar to Angel's semantics, but is elaborated to take into account the particularities of the refinement calculus.

Most of Angel's algebraic laws are not proved. In this work we present their proofs based on the ArcAngel semantics. A normal form is also presented in this work; it is similar to that presented for Angel tactics. Our contribution in this respect is to give more details on the proofs of the lemmas and theorems involved in the strategy of reduction to this normal form.

The constructs of ArcAngel are similar to those of Angel, but are adapted to deal with refinement laws and programs. Moreover, ArcAngel provides structural combinators that are suitable to apply refinement laws to components of programs. Using ArcAngel, we define refinement tactics that embody common development and programming strategies.

Finally, we present Gabriel, a tool support for ArcAngel. Gabriel works as a plug-in to Refine, a tool that semi-automatizes transformations from formal specifications to correct programs with successive refinement laws applications. Gabriel allows its users to create tactics and use them in a program development.

Resumo

O cálculo de refinamentos é uma técnica moderna para o desenvolvimento e implementação de programas de uma maneira precisa, completa e consistente. A partir de uma especificação formal, nós produzimos um programa que implementa corretamente a especificação através de repetidas aplicações de regras de transformação, também chamadas de leis de refinamento. Entretanto, o uso do cálculo de refinamentos pode ser uma tarefa difícil, pois o desenvolvimento de programas pode vir a ser longo e repetitivo.

Estratégias de desenvolvimento são refletidas em sequências de aplicações de leis que são aplicadas repetidamente em desenvolvimentos distintos, ou até mesmo, em pontos diferentes de um mesmo desenvolvimento. A identificação destas táticas de desenvolvimento, documentação, e uso das mesmas em desenvolvimentos de programas como uma simples regra de transformação trazem um grande ganho de tempo e esforço.

Neste trabalho nós apresentamos ArcAngel, uma linguagem para definição de táticas de refinamento baseada em Angel, e formalizamos a sua semântica. Angel é uma linguagem de táticas de propósito geral que assume apenas que regras transformam objetivos de prova. A semântica de ArcAngel é similar a de Angel, mas é elaborada de maneira a levar em consideração as particularidades do cálculo de refinamentos.

A maioria das leis algébricas de Angel não são provadas. Neste trabalho, nós apresentamos suas provas baseadas na semântica de ArcAngel. Também apresentamos neste trabalho uma forma normal; ela é similar àquela apresentada para táticas em Angel. Neste respeito, nossa contribuição é dar mais detalhes nas provas de lemas e teoremas envolvidos na estratégia de redução para esta forma normal.

Os construtores de ArcAngel são similares aos de Angel, mas são adaptados para tratar com leis de refinamento e programas. Além disso, ArcAngel provê combinadores estruturais que são apropriados para aplicar leis de refinamento a componentes de programas. Usando ArcAngel, nós definimos táticas de refinamento que refletem estratégias comuns de desenvolvimento de programas.

Finalmente, nós apresentamos Gabriel, um suporte ferramental para ArcAngel. Gabriel trabalha como um componente de Refine, uma ferramenta que semiautomatiza transformações de espeficações formais para programas corretos através de sucessivas aplicações de leis de refinamento. Gabriel permite aos seus usuários criar táticas e usá-las em desenvolvimento de programas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we motivate the creation of a tactic language for the refinement calculus of Morgan. The motivation for implementing a tool that provides support for the use of this language is also presented. Furthermore, we give a brief introduction to the refinement calculus, and, finally, provide an overview of the whole dissertation.

1.1 Motivation

Software development should include theories and formalisms in order to become software engineering. Most of software developments, however, do not use the already existing theories. This leads to difficulties in developing a relatively low cost trustworthy software, where the time of development is controllable. In [22], Milner affirms that software development theories are as important as computing theories. The experience with the informal techniques is the main reason for using formal methods in the development processes.

The refinement calculus of Morgan [23] is a modern technique for formal program development. By repeatedly applying correctness-preserving transformation rules to an initial specification, we produce a program that implements it. However, applying the refinement calculus may be a hard task, since developments are often long and repetitive. Some development strategies may be captured as sequences of rule applications, and used in different developments, or even several times within a single development. Identifying these strategies [17], documenting them as refinement tactics, and using them as single transformation rules brings a profit in time and effort. Also, a notation for describing derivations can be used for modifying and analyzing formal derivations.

In this work we present a refinement-tactic language called ArcAngel, derived from the more general tactic language, Angel [21, 20]. The flexibility of Angel and its theoretical basis were the reasons for choosing Angel as our basis language. Angel is a general-purpose tactic language that is not tailored to any particular proof tool; it assumes only that rules transform proof goals. A refinement-tactic language must take into account the fact that, when applying refinement laws to a program, we get not only a program, but proof obligations as well. So, the result of applying a tactic is a program and a list of all the proof obligations generated by the individual law applications. We give an informal description of ArcAngel and use it to formalize commonly used refinement strategies. We also provide its formal semantics. Based on these definitions, we have proven over seventy laws of reasoning, which support a strategy of reduction to a normal form of ArcAngel. ArcAngel's constructs are similar to Angel's, but are adapted to deal with the application of refinement laws to programs. In particular, ArcAngel's structural combinators are used to apply tactics to program components.

Other tactic languages can be found in the literature [11, 33, 34, 31, 32, 12, 13, 4, 35, 36, 6, 14]. However, as far as we know, none of them present a formal semantics and a normal form, as ArcAngel does. Furthermore, some of these languages do not present some operators (i.e. recursion and alternation) as ArcAngel does. This limits the power of expression of these languages.

The use of ArcAngel, without a tool support, would still be a hard task. Implementing such a tool would bring further profit in time and effort. We present

Gabriel, a tool support for ArcAngel. Gabriel works as a plug-in to Refine [8] and allows its users to create tactics and use them in a program development.

Some tools for refinement support use existing languages for tactic definition [11, 12, 13, 4, 35, 36, 6, 14]. For this reason, the user must learn complex languages in order to formalize tactics using these tools. Furthermore, some of them have a goal-oriented approach [14]. These tools use existing theorem provers. The refinement consists of proving that the final program implements the initial formal specification. This, however, is not a good approach for program refinement since we do not know the final program from the beginning of the refinement. Finally, some tools do not provide the reuse of previous derivations [31, 32] and do not support procedure constructs [4, 35, 36] as Refine does.

1.2   Refinement Calculus

In this dissertation we assume a previous knowledge of refinement calculus. However, a short description is given in order to establish the notation used in this work.

The refinement calculus is based on a unified language of specification, design and implementation. Its syntax is presented in Figure 1.1. Developing programs consists of repeatedly applying refinement laws to a specification until an adequate program is obtained. All these laws are listed in Appendix C.

A specification has the form $w : [pre, post]$ where w, the frame, lists the variables whose values may change, pre is the precondition, and $post$ is the postcondition. The language used to define the precondition and the postcondition is the predicate calculus. The execution of a specification statement in a state that satisfies the precondition changes the variables listed in the frame so that the final state satisfies the postcondition. If the initial state does not satisfy the precondition, the result cannot be predicted. A precondition $true$ can be omitted. For example, the specification statement $x : [x > 0, x^2 = 1]$ specifies a program which, given x such that $x > 0$, sets x so that $x^2 = 1$.

In the postcondition, 0-subscripted variables can be used to represent the initial value of the corresponding variable. As an example of its use we have $x : [x = x_0 + 1]$. After the execution of this program, the variable x has its value in the initial state incremented by one.

As a development example, we use the following specification statement:

$q, r : [a \geq 0 \wedge b > 0, q = a \text{ div } b \wedge r = a \text{ mod } b]$

It specifies a program which, given a and b, such that $a \geq 0$ and $b > 0$, sets q to the quotient of a divided by b, and sets r to the remaining of this quotient. The

$$
\begin{aligned}
program \quad ::= \quad & name^* : [predicate, predicate] && \textbf{[specification statement]} \\
\mid \quad & name^* := expression^* && \textbf{[multiple} - \textbf{assignment]} \\
\mid \quad & program;\ program && \textbf{[sequence]} \\
\mid \quad & \textbf{if}\ [\!]\ i \bullet predicate \rightarrow program\ \textbf{fi} && \textbf{[conditional]} \\
\mid \quad & \textbf{do}\ [\!]\ i \bullet predicate \rightarrow program\ \textbf{od} && \textbf{[loop]} \\
\mid \quad & [\![\ \textbf{var}\ varDec \bullet program\ ]\!] && \textbf{[variable block]} \\
\mid \quad & [\![\ \textbf{con}\ varDec \bullet program\ ]\!] && \textbf{[constant block]} \\
\mid \quad & [\![\ \textbf{proc}\ name \mathrel{\widehat{=}} procBody \bullet program\ ]\!] && \textbf{[procedure block]} \\
\mid \quad & [\![\ \textbf{proc}\ name \mathrel{\widehat{=}} procBody\ \textbf{variant}\ name\ \textbf{is}\ expression \bullet program\ ]\!] \\
& && \textbf{[variant block]}
\end{aligned}
$$

$$
\begin{aligned}
procBody \quad ::= \quad & program \\
\mid \quad & (parDec \bullet program)
\end{aligned}
$$

$$
\begin{aligned}
parDec \quad ::= \quad & \textbf{val}\ varDec \\
\mid \quad & \textbf{res}\ varDec \\
\mid \quad & \textbf{val-res}\ varDec \\
\mid \quad & parDec;\ parDec
\end{aligned}
$$

$$
\begin{aligned}
varDec \quad ::= \quad & name^+ : Type \\
\mid \quad & varDec;\ varDec
\end{aligned}
$$

Figure 1.1: Abstract Syntax of the Language of Morgan's Refinement Calculus

symbol $\sqsubseteq$ represents the refinement relation such that

$$a$$
$$\sqsubseteq lname(args)$$
$$b$$

represents that, using the law *lname* with arguments *args*, b refines a.

We begin the development by strengthening the postcondition in order to insert the mathematical definition of quotient and of quotient's remaining.

$q, r : [a \geq 0 \wedge b > 0, q = a \text{ div } b \wedge r = a \text{ mod } b]$
$\sqsubseteq strPost((a = q * b + r \wedge 0 \leq r) \wedge \neg\, r \geq b)$
$\qquad q, r : [a \geq 0 \wedge b > 0, (a = q * b + r \wedge 0 \leq r) \wedge \neg\, r \geq b]$ $\qquad\qquad \triangleleft$

This law generates the following proof obligation

$$(a = q * b + r \wedge 0 \leq r) \wedge \neg\, r \geq b \Rightarrow (q = a \text{ div } b \wedge r = a \text{ mod } b).$$

Then, we strengthen again the postcondition in order to introduce the bound limits of the invariant $b > 0$.

$\qquad \sqsubseteq strPost(b > 0 \wedge a = q * b + r \wedge 0 \leq r \wedge \neg\, r \geq b)$
$\qquad\qquad q, r : [a \geq 0 \wedge b > 0, b > 0 \wedge a = q * b + r \wedge 0 \leq r \wedge \neg\, r \geq b]$ $\qquad \triangleleft$

This law generates the following proof obligation

$$(b > 0 \wedge a = q * b + r \wedge 0 \leq r \wedge \neg\, r \geq b) \Rightarrow$$
$$(a = q * b + r \wedge 0 \leq r) \wedge \neg\, r \geq b.$$

We now split the specification into two. We intend to use the first one to initialize the variables used in the iteration generated from the second one.

$\sqsubseteq seqComp(b > 0 \wedge a = q * b + r \wedge 0 \leq r)$
$\qquad q, r : [a \geq 0 \wedge b > 0, b > 0 \wedge a = q * b + r \wedge 0 \leq r];$ $\qquad\qquad\qquad \triangleleft$

$$q, r : \begin{bmatrix} \left(b > 0 \wedge a = q * b + r \wedge 0 \leq r \right), \\ \left(\begin{array}{l} b > 0 \wedge a = q * b + r \\ 0 \leq r \wedge \neg\, r \geq b \end{array} \right) \end{bmatrix} \qquad (i)$$

$\sqsubseteq assign(q, r := 0, a)$
$\qquad q, r := 0, a$

This law generates the following proof obligation

$$a \geq 0 \wedge b > 0 \Rightarrow b > 0 \wedge a = 0 * b + a \wedge 0 \leq a.$$

Then, we introduce the iteration.

(i) $\sqsubseteq iter(\langle r \geq b \rangle, r)$

   do $r \geq b \rightarrow$

$$q, r : \left[\left(\begin{array}{l} b > 0 \wedge a = q * b + r \\ 0 \leq r \wedge r \geq b \\ b > 0 \wedge a = q * b + r \\ 0 \leq r \wedge 0 \leq r < r_0 \end{array} \right), \right] \quad \triangleleft$$

   od

Finally, we introduce the assignment in the body of the iteration. The quotient is incremented by one, and the quotient's remaining is decremented by b.

$\sqsubseteq assignIV(q, r := q + 1, r - b)$

   $q, r := q + 1, r - b$

This law generates the following proof obligation

$$(q = q_0) \wedge (r = r_0) \wedge b > 0 \wedge a = q * b + r \wedge 0 \leq r \wedge r \geq b \Rightarrow$$
$$b > 0 \wedge a = (q + 1) * b + (r - b) \wedge 0 \leq (r - b) \wedge 0 \leq (r - b) < r_0).$$

So, by applying a sequence of refinement laws to the initial specification we get the following executable program.

   $q, r := 0, a;$
   do $r \geq b \rightarrow$
      $q, r := q + 1, r - b$
   od

Besides the specification statement, the language of the refinement calculus includes all the constructors of Dijkstra's language [10]. There are also block constructs to declare local variables, logical constants, and procedures. Variable blocks have the form $[\![\mathbf{var} \ x : T \bullet p]\!]$, where x is a variable name of type T whose scope is restricted to p. Similarly, logical constants c are declared in blocks of the form $[\![\mathbf{con} \ c \bullet p]\!]$.

Procedure blocks are very simple: they begin with the keyword **proc**, the name of the procedure and a program fragment, called the body of the procedure. This body can have argument declarations, if the procedure has arguments. Then, after a $\bullet$, the main program is declared. The general form is $[\![\mathbf{proc} \ name \mathrel{\widehat{=}} body \bullet prog]\!]$.

As an example, we have the program which, given three numbers p, q, and r, exchanges their values such that $p \leq q \leq r$. The specification of this program is

6

presented below.

$$p, q, r : [p \le q \le r \wedge \lfloor p, q, r \rfloor = \lfloor p_0, q_0, r_0 \rfloor]$$

The expression $\lfloor p, q, r \rfloor = \lfloor p_0, q_0, r_0 \rfloor$ assures that the bag containing the values of p, q, and r cannot be changed.

We start the derivation by breaking the program in three assignments. The last assignment sorts p and q. The expression $p \sqcap q$ returns q if $p > q$ and p if $p \le q$. The expression $p \sqcup q$ returns p if $p > q$ and q if $p \le q$.

$\sqsubseteq$ *fassign*$(p, q := (p \sqcap q), (p \sqcup q))$
    $p, q, r : [(p \sqcap q) \le (p \sqcup q) \le r \wedge \lfloor (p \sqcap q), (p \sqcup q), r \rfloor = \lfloor p_0, q_0, r_0 \rfloor];$ $\lhd$
    $p, q := (p \sqcap q), (p \sqcup q)$

The second assignment sorts q and r. Finally, the first assignment also sorts p and q.

$\sqsubseteq$ *fassign*$(q, r := (q \sqcap r), (q \sqcup r))$
    $p, q, r : \left[\begin{array}{l} (p \sqcap (q \sqcap r)) \le (p \sqcup (q \sqcap r)) \le (q \sqcup r) \\ \lfloor (p \sqcap (q \sqcap r)), (p \sqcup (q \sqcap r)), (q \sqcup r) \rfloor = \lfloor p_0, q_0, r_0 \rfloor \end{array} \right];$ $\lhd$
    $q, r := (q \sqcap r), (q \sqcup r)$
$\sqsubseteq$ *assignIV*$(p, q := (p \sqcap q), (p \sqcup q))$
    $p, q := (p \sqcap q), (p \sqcup q)$

This law generates the following proof obligation.

$$p = p_0 \wedge q = q_0 \wedge r = r_0 \Rightarrow$$
$$((p \sqcap q) \sqcap ((p \sqcup q) \sqcap r)) \le ((p \sqcap q) \sqcup ((p \sqcup q) \sqcap r)) \le ((p \sqcup q) \sqcup r)$$
$$\lfloor ((p \sqcap q) \sqcap ((p \sqcup q) \sqcap r)), ((p \sqcap q) \sqcup ((p \sqcup q) \sqcap r)), ((p \sqcup q) \sqcup r) \rfloor =$$
$$\lfloor p_0, q_0, r_0 \rfloor$$

Finally, we introduce the procedure *sort* and, then, we use this procedure in the body of the program.

$\sqsubseteq$ *procNoArgsIntro*$(sort, (p, q := p \sqcap q, p \sqcup q))$
    $[\![$ **proc** $\;sort \mathrel{\widehat{=}} p, q := p \sqcap q, p \sqcup q \;\bullet$
        $p, q := (p \sqcap q), (p \sqcup q);$
        $q, r := (q \sqcap r), (q \sqcup r);$
        $p, q := (p \sqcap q), (p \sqcup q)]\!]$
$\sqsubseteq$ *procNoArgsCall*$()$
    $[\![$ **proc** $\;sort \mathrel{\widehat{=}} p, q := p \sqcap q, p \sqcup q \;\bullet$
        $sort; \; q, r := (q \sqcap r), (q \sqcup r); \; sort]\!]$

So, our derivation results in the following program. This program defines the

procedure *sort* and use it to sort p, q, and r.

$$\llbracket\, \textbf{proc}\;\; sort \;\widehat{=}\; p,q := p \sqcap q, p \sqcup q \bullet$$
$$sort;\; q,r := (q \sqcap r),(q \sqcup r);\; sort \rrbracket$$

The arguments can be passed by value, using the keyword **val**, by result, using the keyword **res**, or by value-result, using the keyword **val-res**. An example of a parameterized procedure is $\llbracket\, \textbf{proc}\;\; inc \;\widehat{=}\; (\textbf{val-res}\; n : \mathbb{N} \bullet n := n+1) \bullet inc(n) \rrbracket$. The body of the procedure in this case is a parameterized command [2].

For example, we have the square root program. Its specification is

$$x : [0 \leq x, x^2 = x_0]$$

We start the development by introducing the procedure *sqrts*, which returns the square root of a given real number.

$\sqsubseteq\;\; procArgsIntro(sqrts, b : [0 \leq a, b^2 = a], (\textbf{val}\;\; a : \mathbb{R};\; \textbf{res}\;\; b : \mathbb{R}))$
  $\llbracket\, \textbf{proc}\;\; sqrts \;\widehat{=}\; (\textbf{val}\;\; a : \mathbb{R};\; \textbf{res}\;\; b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a]) \bullet$
    $x : [0 \leq x, x^2 = x_0]$ ◁
  $\rrbracket$

The next step is to introduce a substitution by value and contract the frame to remove the variable a.

$\sqsubseteq\;\; callByValueIV(x, a)$
  $(\textbf{val}\;\; a : \mathbb{R} \bullet x, a : [0 \leq a, x^2 = a_0])(x)$ ◁
$\sqsubseteq\;\; contractFrame(a)$
  $(\textbf{val}\;\; a : \mathbb{R} \bullet x : [0 \leq a, x^2 = a])(x)$ ◁

Then, we introduce a substitution by result and use the law *multiArgs* to put our two kinds of substitution together.

$\sqsubseteq\;\; callByResult(x, b)$
  $(\textbf{res}\;\; b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a])(x)$ ◁
$\sqsubseteq\;\; multiArgs()$
  $(\textbf{val}\;\; a : \mathbb{R}, \textbf{res}\;\; b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a])(x, x)$ ◁

Finally, we introduce the procedure call.

$\sqsubseteq\;\; procArgsCall()$
  $sqrts(x, x)$

Our resulting program introduces the procedure *sqrts* and uses it to calculate

the square root of a given number x.

$$[\![\ \textbf{proc}\ \ sqrts \mathrel{\widehat{=}} (\textbf{val}\ \ a : \mathbb{R}, \textbf{res}\ \ b : \mathbb{R} \bullet b : [0 \le a, b^2 = a]) \bullet$$
$$sqrts(x, x)$$
$$]\!]$$

Variant blocks are used to develop recursive procedures. Besides declaring the procedure and the main program, a variant block declares a variant e named v used to develop a recursive implementation for the procedure. The general form is $[\![\ \textbf{proc}\ \ name \mathrel{\widehat{=}} body\ \textbf{variant}\ v\ \textbf{is}\ e \bullet prog]\!]$.

We present the factorial program as an example of a recursive procedure development. We have the following specification as the initial specification.

$f : [f = n! * 1]$ ◁

The variant introduction is the first step of the development. We define the value of the variable m as the variant V.

$\sqsubseteq\ variantIntro(fact, V, m, (f : [f = m! * k]), \textbf{val}\ \ m, k : \mathbb{N})$
$\quad [\![\ \textbf{proc}\ \ fact \mathrel{\widehat{=}} (\textbf{val}\ \ m, k : \mathbb{N} \bullet f : [V = m, f = m! * k])$
$\quad\quad \textbf{variant}\ \ V\ \textbf{is}\ m \bullet$
$\quad\quad f : [f = n! * 1]$ ◁
$\quad]\!]$

Then we introduce a substitution by value of the variables m and k.

$\sqsubseteq\ callByValue(m, k : \mathbb{N}, \langle n, 1 \rangle)$
$\quad [\![\ \textbf{proc}\ \ fact \mathrel{\widehat{=}} (\textbf{val}\ \ m, k : \mathbb{N} \bullet f : [V = m, f = m! * k])$
$\quad\quad \textbf{variant}\ \ V\ \textbf{is}\ m \bullet$
$\quad\quad (\textbf{val}\ \ m, k : \mathbb{N} \bullet f : [f = m! * k])(n, 1)$
$\quad]\!]$ ◁

The next step is to introduce the procedure call in the variant block.

$\sqsubseteq\ procVariantBlockCall()$
$\quad [\![\ \textbf{proc}\ \ fact \mathrel{\widehat{=}} (\textbf{val}\ \ m, k : \mathbb{N} \bullet f : [V = m, f = m! * k])$ ◁
$\quad\quad \textbf{variant}\ \ V\ \textbf{is}\ m \bullet$
$\quad\quad fact(n, 1)$
$\quad]\!]$

Now, we start to develop the body of the procedure $fact$. First, we introduce an

alternation in order to check if we must stop the recursion $(m = 0)$ or not $(m > 0)$.

$\sqsubseteq$ **law** $alt(\langle m = 0, m > 0 \rangle)$
    if $m = 0 \rightarrow f : [V = m \wedge m = 0, f = m! * k]$ $\lhd$
        $[\!]\ m > 0 \rightarrow f : [V = m \wedge m > 0, f = m! * k]$ (i)
    fi

This law application generates the following proof obligation

$$V = m \Rightarrow m = 0 \vee m > 0$$

The base case is when we have m equals to 0. In such case, k must receive the value of the variable f.

$\sqsubseteq$ $assign(f := k)$
    $f := k$

The following proof obligation is generated in this case

$$V = m \wedge m = 0 \Rightarrow k = 0! * k$$

If $m > 0$ we should make a recursive call. We first strengthen the postcondition in order to make a recursive call of procedure $fact$.

$(i) \sqsubseteq$ $strPost(f = (m - 1)! * m * k)$
    $f : [V = m \wedge m > 0, f = (m - 1)! * m * k]$ $\lhd$

We have the following proof obligation

$$f = (m - 1)! * m * k \Rightarrow f = m! * k$$

Then, we introduce a substitution by value and weaken the pre condition.

$\sqsubseteq$ $callByValue(\langle m, k \rangle, \langle m - 1, m * k \rangle)$
    $(\textbf{val}\ m, k : \mathbb{N} \bullet f : [V = m + 1 \wedge m + 1 > 0, f = m! * k])$
        $(m - 1, m * k)$ $\lhd$
$\sqsubseteq$ $weakPre(0 \leq m < V)$
    $f : [0 \leq m < V, f = m! * k])$ $\lhd$

This law generates the following proof obligation

$$V = m + 1 \wedge m + 1 > 0 \Rightarrow 0 \leq m < V$$

At this point of the program development, we have the following collected pro-

gram.

$$\llbracket \textbf{proc} \quad fact \mathrel{\widehat{=}} (\textbf{val} \quad m, k : \mathbb{N} \bullet$$
$$\textbf{if} \quad m = 0 \rightarrow f := k$$
$$\llbracket \quad m > 0 \rightarrow f : [0 \leq m < V, f = m! * k]$$
$$\textbf{fi})$$
$$\textbf{variant} \quad V \textbf{ is } m \bullet$$
$$fact(n, 1)\rrbracket$$

Finally, we refine this program by introducing a recursive call to the procedure *fact* as seen below.

$$\sqsubseteq \quad recursiveCall()$$
$$\llbracket \textbf{proc} \quad fact \mathrel{\widehat{=}} (\textbf{val} \quad m, k : \mathbb{N} \bullet$$
$$\textbf{if} \quad m = 0 \rightarrow f := k$$
$$\llbracket \quad m > 0 \rightarrow fact(m - 1, m * k)$$
$$\textbf{fi})$$
$$\textbf{variant} \quad V \textbf{ is } m \bullet$$
$$fact(n, 1)\rrbracket$$

This law application generates the following proof obligation

$$f : [V = n, f = m! * k]$$
$$\sqsubseteq$$
$$\textbf{if} \quad m = 0 \rightarrow f := k$$
$$\llbracket \quad m > 0 \rightarrow f : [0 \leq m < V, f = m! * k]$$
$$\textbf{fi}$$

This finishes the development of the factorial program.

The development of some examples seen here make use of development strategies which can be found in the literature. For instance, the first example introduces an initialized iteration using a conjunction as its invariant. The sequence of law applications to introduce such an iteration can be formalized as a tactic, reducing time and effort in program developments. Next chapter presents ArcAngel, a language for tactic formalization.

1.3 Overview

Chapter 2 introduces ArcAngel. First, we present ArcAngel's Syntax and give an informal description of the language. Then, we present some commonly used refinement strategies (tactics) written in ArcAngel. Finally, we present ArcAngel's formal semantics.

Based on this semantics, Chapter 3 presents laws of reasoning. These laws are used to support a strategy of reduction to a normal form for ArcAngel. All the laws used in this work are proved in Appendix D.

In Chapter 4 we discuss the implementation of Gabriel, a tool that supports the use of ArcAngel. We present the concepts of the tool. Then we discuss briefly its user interface, presenting the constructs of ArcAngel available in Gabriel, and an example of a tactic creation and usage.

Finally, in Chapter 5 we summarize the results obtained and make some final considerations. Related and future works are also discussed in this chapter.

Chapter 2

ArcAngel

In this chapter we present the language ArcAngel, a refinement-tactic language based on Angel. Most of basic tactics in ArcAngel are inherited from Angel. However, almost all ArcAngel's structural combinators are not defined in Angel. ArcAngel's extension for Angel can be found in Appendix A. All the program development strategies in the literature are formalized as tactics written in ArcAngel and their use are also presented. Finally, we present ArcAngel's formal semantics which is totally different from Angel's formal semantics.

2.1   **ArcAngel**'s Syntax

The syntax of ArcAngel is displayed in Figure 2.1. The syntactic category *args* is the set of (possibly empty) comma-separated lists of arguments enclosed in parentheses. An argument is a predicate; an expression; a variable or constant declaration; a list of predicates, expressions, variables, or arguments declarations; or a program. The category *pars* is the set of possibly empty comma-separated lists of (parameter) names enclosed in parentheses. The definitions of these syntactic categories are standard and are omitted. Finally, the notation $tactic^+$ is used to represent a non-empty list of tactics; in examples, we number the elements of this list. A pair of square brackets represents optional clauses.

There are three distinct kinds of tactics: basic tactics are the simplest tactics; tacticals are combination of tactics; and structural combinators are tactics used to handle parts of a program. Some of the basic tactics and most of the tacticals are original to ArcAngel; most of the structural combinators in ArcAngel do not exist in Angel. ArcAngel's extension for Angel can be found in Appendix A.

2.1.1   Basic Tactics

The most basic tactic is a simple law application: **law** $n(a)$. The application of this tactic to a program has two possible outcomes. If the law n with arguments a is applicable to the program, then the application actually occurs and the program is changed, possibly generating proof obligations. If the law is not applicable to the program, then the application of the tactic fails.

The construct **tactic** $n(a)$ applies a previously defined tactic as though it were a single law. The trivial tactic **skip** always succeeds, and the tactic **fail** always fails; neither generates any proof obligations. The tactic **abort** neither succeeds nor fails, but runs indefinitely.

2.1.2   Tacticals

In ArcAngel, tactics may be sequentially composed: t_1; t_2. This tactic first applies t_1 to the program, and then applies t_2 to the outcome of the application of t_1. If either t_1 or t_2 fails, then so does the whole tactic. When it succeeds, the proof obligations generated by the application of this tactic are those resulting from the application of t_1 and t_2.

For example, consider the program $x : [\, x \geq 1 \,]$. We could implement this by first strengthening the postcondition to $x = 1$, and then replacing it with a simple assignment. The necessary tactic is **law** $strPost(x = 1)$; **law** $assign(x := 1)$. After the application of the first tactic, the resulting program is $x : [\, x = 1 \,]$ and the proof obligation is $x = 1 \Rightarrow x \geq 1$. After the application of the assignment introduction

$$
\begin{array}{lll}
\textit{tactic} \quad ::= & \textbf{law } \textit{name args} & \textbf{[law application]} \\
& \mid \quad \textbf{tactic } \textit{name args} & \textbf{[tactic application]} \\
& \mid \quad \textbf{skip} \mid \textbf{fail} \mid \textbf{abort} & \\
& \mid \quad \textit{tactic; tactic} & \textbf{[sequence]} \\
& \mid \quad \textit{tactic} \mid \textit{tactic} & \textbf{[alternation]} \\
& \mid \quad ! \, \textit{tactic} & \textbf{[cut]} \\
& \mid \quad \mu \, \textit{name} \bullet \textit{tactic} & \textbf{[recursion]} \\
& \mid \quad \textbf{succs } \textit{tactic} \mid \textbf{fails } \textit{tactic} & \textbf{[assertions]} \\
& \mid \quad \textit{tactic} \,\fbox{;}\, \textit{tactic} & \textbf{[structural combinators]} \\
& \mid \quad \fbox{if}\, \textit{tactic}^{+} \,\fbox{fi} \mid \fbox{do}\, \textit{tactic}^{+}\, \fbox{od} & \\
& \mid \quad \fbox{var}\, \textit{tactic}\, \fbox{]\!]} \mid \fbox{con}\, \textit{tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pmain}\, \textit{tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pmainvariant}\, \textit{tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pbody}\, \textit{tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pbodyvariant}\, \textit{tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pbodymain}\, \textit{tactic tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{pmainvariantbody}\, \textit{tactic tactic}\, \fbox{]\!]} & \\
& \mid \quad \fbox{val}\, \textit{tactic} & \\
& \mid \quad \fbox{res}\, \textit{tactic} & \\
& \mid \quad \fbox{val-res}\, \textit{tactic} & \\
& \mid \quad \fbox{parcommand}\, \textit{tactic} & \\
& \mid \quad \textbf{con } v \bullet \textit{tactic} & \textbf{[constants]} \\
& \mid \quad \textbf{applies to } \textit{program} \, \textbf{do} \, \textit{tactic} & \textbf{[patterns]} \\
\\
\textit{tacDec} \quad ::= & \textbf{Tactic } \textit{name pars tactic} & \\
& [\,\textbf{proof obligations } \textit{predicate}^{+}\,] & \\
& [\,\textbf{program generated } \textit{program}\,]\, \textbf{end} & \\
\\
\textit{tacProg} \quad ::= & \textit{tacDec}^{*}\textit{tactic} &
\end{array}
$$

Figure 2.1: Abstract Syntax of ArcAngel

15

law, we get the program $x := 1$ with the two proof obligations $x = 1 \Rightarrow x \geq 1$ and $true \Rightarrow 1 = 1$. The discharge of proof obligations is not tackled by ArcAngel. In our examples, and in Gabriel, the generated proof obligations are listed, but not proved. Support for their discharge is left as future work.

In examples, for clarity, sometimes we use assignments as arguments as a syntactic sugar. For instance, above, we write **law** $assign(x := 1)$, but the arguments are actually x and 1. We use the same sort of notation for arguments and parameters of tactics.

Tactics may also be combined in alternation: $t_1 \mid t_2$. First t_1 is applied to the program. If the application of t_1 leads to success, then the composite tactic succeeds; otherwise t_2 is applied to the program. If the application of t_2 leads to success then the composite tactic succeeds; otherwise the composite tactic fails. If one of the tactics aborts, the whole tactic aborts. When a tactic contains many choices, the first choice that leads to success is selected. It is the angelic nature of this nondeterminism, in which alternative tactics that lead to success are chosen over those that lead to failure, that earned Angel and ArcAngel (A Refinement Calculus for Angel) their names.

For example, suppose that we have a useful tactic t that relies on the frame containing only the variable x, and that we want to generalize it to t', which applies to specification statements with frame x, y. We could define t' to be **law** $contractFrame(y); \; t$. Unfortunately, the resulting tactic no longer applies where we found t to be useful, since contracting the frame will surely fail in such cases. Instead, we can use the compound tactic $(\textbf{law } contractFrame(y) \mid \textbf{skip}); \; t$. Now, if contracting the frame works, we do it; if it does not, then we ignore it; either way, we apply t next. The tactic $(\textbf{skip} \mid \textbf{law } contractFrame(y)); \; t$ has the same effect. In this case, the tactic t is applied without any prior change to the specification statement. If this application does not succeed, the law $contractFrame$ is applied before a new attempt to apply t.

The angelic nondeterminism can be implemented through backtracking: in the case of failure, law applications are undone to go back to the last point where further alternatives are available and can be explored. This, however, may result in inefficient searches. Some control over this is given to the programmer through the cut operator. The cut tactic $!t$ behaves like t, except that it returns the first successful application of t. If a subsequent tactic application fails, then the whole tactic fails. Consider our previous example once more, supposing that **law** $contractFrame(y)$ succeeds, but that t subsequently does not. There is no point is applying the trivial tactic **skip** and then trying t again. Instead, we should cut [7] the search: $!(\textbf{law } contractFrame(y) \mid \textbf{skip}); \; t$.

Suppose that we have a tactic u that performs some simple task, like identifying an equation in a specification postcondition and then applying the rule of *following-assignment*. Such a simple tactic may be made more useful by applying

it repeatedly, until it can be applied no more. ArcAngel has a fixed-point operator that allows us to define recursive tactics. Using this operator, we can define a tactic that applies u exhaustively: the tactic $\mu\,X\,\bullet\,(\,u;\ X\,\mid\,\mathbf{skip}\,)$ applies u as many times as possible, terminating with success when the application of u fails. Recursive application of a tactic may lead to nontermination, in which case the result is the same as the trivial tactic **abort**.

The tactic $\mathbf{con}\,v\,\bullet\,t$ introduces v as a set of free variables ranging over appropriate syntactic classes in t, angelically chosen so that as many choices as possible succeed. The tactic **applies to** p **do** t is used to define a pattern for the programs to which the tactic t can be applied. It introduces a meta-program p that characterizes the programs to which this tactic is applicable; the meta-variables used in p can then be used in t. For example, the meta-program $w : [\,pre, post_1 \vee post_2\,]$ characterizes those specifications whose postcondition is a disjunction; here, pre, $post_1$, and $post_2$ are the meta-variables. Consider as an example a commonly used refinement tactic: strengthening a postcondition by dropping a disjunct. This tactic is formalized as **applies to** $w : [\,pre, post_1 \vee post_2\,]$ **do law** $strPost(post_1)$.

Two tactics are used to make tactic assertions that check the outcome of applying a tactic. The tactic **succs** t fails whenever t fails, and behaves like **skip** whenever t succeeds. On the other hand, **fails** t behaves like **skip** if t fails, and fails if t succeeds. If the application of t runs indefinitely, then these tacticals behave like **abort**. A simple example is a test to see whether a program is a specification statement. We know that it is always possible (but seldom desirable) to strengthen a specification statement's postcondition to *false*; however, the tactic applies only to specification statements. So, our test may be coded as **succs**(**law** $strPost(false)$).

2.1.3 Structural Combinators

Very often, we want to apply individual tactics to subprograms. The tactic $t_1 \;\fbox{;}\; t_2$ applies to programs of the form $p_1;\ p_2$. It returns the sequential composition of the programs obtained by applying t_1 to p_1 and t_2 to p_2; the proof obligations generated are those arising from both tactic applications. Combinators like $\fbox{;}$ are called *structural combinators*. These combinators correspond to the syntactic structures in the programming language. Essentially, there is one combinator for each syntactic construct.

For alternation, there is the structural combinator $\fbox{if}\,t_1\,\fbox{[]}\,\ldots\,\fbox{[]}\,t_n\,\fbox{fi}$, which applies to an alternation $\mathbf{if}\ g_1\,\rightarrow\,p_1\ [\!]\ \ldots\ [\!]\ g_n\,\rightarrow\,p_n\ \mathbf{fi}$. It returns the result of applying each tactic t_i to the corresponding program p_i. For example, if we have

17

the program

> **if** $a < b \to x : [\, x < 0\,] [\!] a = b \to x : [\, x = 0\,] [\!] a > b \to x : [\, x > 0\,]$ **fi**

and tactic

> $\boxed{\textbf{if}}\,\textbf{law}\;assign(x := -1)\;\boxed{[\!]}\,\textbf{law}\;assign(x := 0)\;\boxed{[\!]}\,\textbf{law}\;assign(x := 1)\;\boxed{\textbf{fi}}$

we obtain three proof obligations $true \Rightarrow -1 < 0$ and $true \Rightarrow 0 = 0$, and $true \Rightarrow 1 > 0$, and **if** $a < b \to x := -1$ $[\!]$ $a = b \to x := 0$ $[\!]$ $a > b \to x := 1$ **fi** as the resulting program. For iterations **do** $g_1 \to p_1$ $[\!]$ $\ldots$ $[\!]$ $g_n \to p_n$ **od**, we have a similar structural combinator $\boxed{\textbf{do}}\,t_1\,\boxed{[\!]}\,\ldots\,\boxed{[\!]}\,t_n\,\boxed{\textbf{od}}$.

The structural combinator $\boxed{\textbf{var}}\,t\,\boxed{]\!]}$ applies to a variable block, and $\boxed{\textbf{con}}\,t\,\boxed{]\!]}$ to a logical constant block; each applies its tactic t to the body of the block. For example, if we apply to $[\![\,\textbf{var}\;x : \mathbb{N} \bullet x : [\, x \geq 0\,]\,]\!]$ the structural combinator $\boxed{\textbf{var}}\;\textbf{law}\;assign(x := 10)\,\boxed{]\!]}$, we get $[\![\,\textbf{var}\;x : \mathbb{N} \bullet x := 10\,]\!]$ and the proof obligation $true \Rightarrow 10 \geq 0$.

In the case of procedure blocks and variant blocks, the structural combinators $\boxed{\textbf{pmain}}\,t\,\boxed{]\!]}$ and $\boxed{\textbf{pmainvariant}}\,t\,\boxed{]\!]}$ are used, respectively; they apply t to the main program of the blocks. For example, applying $\boxed{\textbf{pmain}}\;\textbf{law}\;assign(x := 10)\,\boxed{]\!]}$ to the procedure block

> $[\![\,\textbf{proc}\;nonNeg \mathrel{\widehat=} x : [\, x > 0\,] \bullet x : [\, x \geq 0\,]\,]\!]$

we get $[\![\,\textbf{proc}\;nonNeg \mathrel{\widehat=} x : [\, x > 0\,] \bullet x := 10\,]\!]$ and the proof obligation $true \Rightarrow 10 \geq 0$.

To apply a tactic to a procedure body, we use the structural combinators $\boxed{\textbf{pbody}}\,t\,\boxed{]\!]}$ and $\boxed{\textbf{pbodyvariant}}\,t\,\boxed{]\!]}$, which apply to procedure and variant blocks, respectively. For example, if we apply the tactic

> $\boxed{\textbf{pbody}}\;\textbf{law}\;assign(x := 10)\,\boxed{]\!]}$

to the program

> $[\![\,\textbf{proc}\;\;nonNeg \mathrel{\widehat=} x : [x \geq 0] \bullet nonNeg]\!]$

we get

> $[\![\,\textbf{proc}\;\;nonNeg \mathrel{\widehat=} x := 10 \bullet nonNeg]\!]$

and the proof obligation $true \Rightarrow 10 \geq 0$.

It is also possible to apply tactics to a procedure body and to the main program of a procedure block, or a variant block, at the same time. For this. we use the structural combinators $\boxed{\textbf{pbodymain}}\,t_b\;t_m\,\boxed{]\!]}$ and $\boxed{\textbf{pmainvariantbody}}\,t_b\;t_m\,\boxed{]\!]}$, which apply to procedure blocks and variant blocks, respectively. They apply t_b to the body of the procedure, and t_m to the main program.

For argument declaration, the combinators $\boxed{\textbf{val}}\,t$, $\boxed{\textbf{res}}\,t$, and $\boxed{\textbf{val-res}}\,t$ are used, depending on whether the arguments are passed by value, result, or value-result.

For example, when the following tactic

$$\boxed{\textbf{pbody}}\,\boxed{\textbf{val-res}}\ \textbf{law}\ \textit{assign}(x := 10)\boxed{]\!]}$$

is applied to the procedure block

$$[\![\ \textbf{proc}\ \textit{nonNegArg} \mathrel{\widehat{=}} (\textbf{val-res}\ x : \mathbb{N} \bullet x : [\,x \geq 0\,]) \bullet x : [\,x \geq 0\,]\,]\!]$$

it returns the proof obligation $\textit{true} \Rightarrow 10 \geq 0$ and the program

$$[\![\ \textbf{proc}\ \textit{nonNegArg} \mathrel{\widehat{=}} (\textbf{val-res}\ x : \mathbb{N} \bullet x := 10 \bullet x : [\,x \geq 0\,]\,]\!]$$

Sometimes, however, we are not concerned with the type of argument declaration. In these cases, we use the structural combinator $\boxed{\textbf{parcommand}}\,t$. For example, if we apply the tactic

$$\boxed{\textbf{pmain}}\,\boxed{\textbf{parcommand}}\ \textbf{law}\ \textit{assign}(x := 10)\boxed{]\!]}$$

to the program

$$[\![\ \textbf{proc}\ \textit{nonNegArg} \mathrel{\widehat{=}} \textit{body} \bullet (\textbf{val-res}\ x : \mathbb{N};\ \textbf{val}\ y : \mathbb{N} \bullet x : [\,x \geq 0\,])\,]\!]$$

it returns the proof obligation $\textit{true} \Rightarrow 10 \geq 0$ and the program

$$[\![\ \textbf{proc}\ \textit{nonNegArg} \mathrel{\widehat{=}} \textit{body} \bullet (\textbf{val-res}\ x : \mathbb{N};\ \textbf{val}\ y : \mathbb{N} \bullet x := 10)\,]\!]$$

We may declare a named tactic with arguments using **Tactic** $n(a)\,t$ **end**. For documentation purposes, we may include the clause **proof obligations** and the clause **program generated**; the former lists the proof obligations generated by the application of t, and the latter shows the program generated. These two clauses are optional as this information can be inferred from the tactic itself. The effect of **Tactic** $n(a)\,t$ **end** is that of t, which is named n and uses the arguments a; the presence of the optional clauses does not affect the behavior.

A tactic program consists of a sequence of tactic declarations followed by a tactic that usually makes use of the declared tactics.

2.2   Examples

In this section we give a few examples of ArcAngel programs. The first two, *followingAssign* and *leadingAssign*, implement two derived rules from Morgan's calculus. The other tactics formalize all the refinement strategies we found in the literature. Some of these examples were first presented in [28]. More examples can also be found in [25].

2.2.1 Following Assignment

In [23, p.32], the derived rule *following assignment* is presented as a useful combination of the *assignment* and *sequential composition* laws; in other words, it is rather like a tactic for using the more basic laws of the calculus. Of course, there is a big difference between a tactic and a derived rule. The former is a program that applies rules; proof obligations arise from the law applications. A derived rule is a law itself, that is proved in terms of applications of other laws. On the other hand, tactics are much more flexible than derived rules, since they can make choices about the form of the resulting program; derived rules cannot.

Following assignment splits a specification statement into two pieces, the second of which is implemented by the assignment $x := E$. If we apply to $w, x : [\,pre, post\,]$ the tactic **law** $seqComp(\,post[x \setminus E]\,)$, we get the program

$$w, x : [\,pre, post[x \setminus E]\,];\ w, x : [\,post[x \setminus E], post\,]$$

All the refinement laws are defined in Appendix C.

If we now apply $assign(\,x := E\,)$ to the second statement, we get our assignment $x := E$ and the proof obligation $post[x \setminus E] \Rightarrow post[x \setminus E]$. This is a simple tautology, but remains part of the documentation of the tactic, as the proof obligation is raised every time the tactic is applied.

> **Tactic** $followingAssign\,(\,x, E\,)$
>   **applies to** $w, x : [\,pre, post\,]$ **do**
>
>   **law** $seqComp(\,post[x \setminus E])$; (**skip** $\boxed{;}$ **law** $assign(\,x := E\,)$)
>
>   **proof obligations**
>
> $post[x \setminus E] \Rightarrow post[x \setminus E]$
>
>   **program generated**
>
> $w, x : [\,pre, post[x \setminus E]\,];$
> $x := E$
>
> **end**

There is a further restriction on the use of this tactic: it uses a simple form of the law of sequential composition that forbids initial variables in post conditions. If applied to a specification statement that does not satisfy this restrictions, *followingAssign* fails.

2.2.2 Leading Assignment

In [23, p.71], the presentation of the derived rule for *leading assignment* is a little more complicated than that of the *following assignment*:

Law 13.1 § *leading assignment* For any expression E,

$$w, x : [\, pre[x \setminus E], post[x_0 \setminus E_0]\,]$$

$$\sqsubseteq \quad \begin{array}{l} x := E; \\ w, x : [\, pre, post\,] \end{array}$$

The expression E_0 is that obtained from E by replacing all free occurrences of x and w with x_0 and w_0, respectively. The complication arises from the need to find predicates *pre* and *post*, such that, when appropriate substitutions are made, they match the current goal.

Tactic $leadingAssign\,(\,x, X, E\,)$
    applies to $\ w, x : [\, pre[x \setminus E], post[x_0 \setminus E_0]\,]$ **do**

        law $seqCompCon(\, pre \wedge x = E_0, X, x\,)$;
        $\boxed{\textbf{con}}$ **law** $assignIV(\, x := E\,)\boxed{;}$

            law $strPostIV(post)$; **law** $weakPre(pre)\,\boxed{]\!]}$;

        law $removeCon(X)$

    proof obligations

        $(x = x_0) \wedge pre\,[x \setminus E] \Rightarrow (\,pre \wedge x = E[x \setminus x_0])[x \setminus E]$,
        $(\,pre \wedge x = E\,[x \setminus X])[x, w \setminus x_0, w_0] \wedge post \Rightarrow post[x_0 \setminus E_0][x_0 \setminus X]$,
        $(\,pre \wedge x = E\,[x \setminus X]) \Rightarrow pre$

    program generated

        $x := E;\ w, x : [\, pre, post\,]$
    end

If applied to a specification statement $w, x;\ [pre[x \setminus E], post[x_0 \setminus E_0]]$, this tactic first splits it using the law *seqCompCon*. As *seqComp*, this law introduces sequences, but it applies to specification statements that make use of initial variables; it declares a logical constant to record the initial value of x. This gives:

$[\![\, \textbf{con}\, X\, \bullet$
    $x : [pre\,[x \setminus E], pre \wedge x = E\,[x \setminus x_0]\,]$;
    $w, x : [pre \wedge x = E[x \setminus X], post[x_0 \setminus E_0][x_0 \setminus X]\,]$
$]\!]$

The first specification statement is refined to $x := E$, using the *assignIV* law: a

law for introducing assignments that considers the presence of initial variables. The precondition and postcondition of the second specification statement are changed to *pre* and *post*, respectively, and finally the logical constant is removed. The law *strPostIV* is the strengthening postcondition law that takes initial variables into account. The proof obligations are generated by the applications of the laws *assignIV*, *strPostIV*, and *weakPre*, respectively; they always hold.

2.2.3   TakeConjAsInv

This refinement strategy that we formalize in this section aims at the development of an initialized iteration. It consists of taking a conjunct of the postcondition of the program specification as the main part of the invariant. The tactic *takeConjAsInv* defined below follows this strategy to transform a specification $w : [pre, invconj \land notGuard]$ into an initialized iteration.

> **Tactic** $takeConjAsInv$ ($invBound, (ivar := ival), variant$)
> **applies to** $w : [\, pre, invConj \land \neg\ guard \,]$ **do**
>
> >   **law** $strPost(\, invBound \land invConj \land \neg\ guard\,);$
> >   **law** $seqComp(\, invBound \land invConj\,);$
> >   (**law** $assign(\, ivar := ival\,)\,\boxed{;}\,$ **law** $iter(\langle guard \rangle, variant))$
>
> >   **proof obligations**
>
> > > $invBound \land invConj \land \neg\ guard \Rightarrow invConj \land \neg\ guard,$
> > > $pre \Rightarrow (\, invBound \land invConj\,)[ivar \setminus ival]$
>
> >   **program generated**
>
> > > $ivar := ival;$
> > >   **do** $guard \rightarrow$
> > >   > $w : [\, invBound \land invConj \land guard,$
> > >   >   > $invBound \land invConj \land 0 \le variant < variant_0\,]$
> > >   **od**
>
> **end**

This tactic has three arguments: a predicate *invBound*, an assignment $ivar := ival$, where *ivar* is a list of variables and *ival* is an equal-length list of values, and an integer expression *variant*. This tactic applies to a specification statement $w : [pre, invConj \land \neg\ guard]$ to introduce an initialized iteration whose invariant is $invConj \land invBound$ and whose variant is *variant*. The initialization is $ivar := ival$. Typically, the predicate *invBound* states the range limits of indexing variables of the iteration. The conjunction $invConj \land invBound$ is used as invariant of the iteration.

The tactic *takeConjAsInv* first strengthens the postcondition (law *strPost*) using the argument *invBound*, then it introduces a sequential composition (law *seqComp*). Afterwards, the law *assign* is applied to the first program of the composition to derive the initialization, and the law *iter* is applied to the second program in order to introduce the iteration. The first proof obligation is generated by the application of the law *strPost*, and the second by the application of the law *assign*.

Example.   We now present the use of a tactic in the first example presented in Section 1.2. This example develops a program which, given a and b, such that $a \geq 0$ and $b > 0$, sets q to the quotient of a divided by b, and sets r to the remaining of this quotient.

As already presented before, we begin the development by strengthening the postcondition in order to insert the mathematical definition of quotient and of a quotient's remaining.

$q, r : [a \geq 0 \land b > 0, q = a \text{ div } b \land r = a \text{ mod } b]$
$\sqsubseteq strPost((a = q * b + r \land 0 \leq r) \land \neg \, r \geq b)$
$\qquad q, r : [a \geq 0 \land b > 0, (a = q * b + r \land 0 \leq r) \land \neg \, r \geq b]$ $\qquad\qquad \lhd$

This law generates the following proof obligation.

$$(a = q * b + r \land 0 \leq r) \land \neg \, r \geq b \Rightarrow (q = a \text{ div } b \land r = a \text{ mod } b).$$

Now, we apply the tactic *takeConjAsInv* using the bound limits of the invariant $b > 0$, the initialization of the variables $q, r := 0, a$ and the variant r of the iteration.

$\sqsubseteq takeConjAsInv(b > 0, (q, r := 0, a), r)$

This application represents a simple application and should give a program as result. However, for better understanding, we present the details of tactics applications. We follow with the application of the tactic *takeConjAsInv*.

$\qquad \sqsubseteq strPost(b > 0 \land a = q * b + r \land 0 \leq r \land \neg \, r \geq b)$
$\qquad\qquad q, r : [a \geq 0 \land b > 0, b > 0 \land a = q * b + r \land 0 \leq r \land \neg \, r \geq b]$ $\qquad \lhd$

This law generates the following proof obligation.

$$(b > 0 \land a = q * b + r \land 0 \leq r \land \neg \, r \geq b) \Rightarrow$$
$$(a = q * b + r \land 0 \leq r) \land \neg \, r \geq b.$$

Most of the proof obligations generated in the examples are very simple like this one; they follow directly from the properties of the predicate calculus and of the data type involved, mainly numbers. Then, the tactic splits the specification into

two parts. The first one is refined to the initialization of the iteration.

$$\sqsubseteq seqComp(b > 0 \land a = q * b + r \land 0 \le r)$$
$$q, r : [a \ge 0 \land b > 0, b > 0 \land a = q * b + r \land 0 \le r]; \qquad \lhd$$
$$q, r : \left[\begin{pmatrix} b > 0 \land a = q * b + r \land 0 \le r \end{pmatrix}, \\ \begin{pmatrix} b > 0 \land a = q * b + r \\ 0 \le r \land \neg\, r \ge b \end{pmatrix} \right] \qquad (i)$$
$$\sqsubseteq assign(q, r := 0, a)$$
$$q, r := 0, a$$

This law generates the following proof obligation.

$$a \ge 0 \land b > 0 \Rightarrow b > 0 \land a = 0 * b + a \land 0 \le a.$$

Finally, the tactic introduces the iteration.

$$(i) \sqsubseteq iter(\langle r \ge b \rangle, r)$$
$$\textbf{do}\ \ r \ge b \rightarrow$$
$$q, r : \left[\left\{ \begin{matrix} b > 0 \land a = q * b + r \\ 0 \le r \land r \ge b \\ b > 0 \land a = q * b + r \\ 0 \le r \land 0 \le r < r_0 \end{matrix} \right\}, \right] \qquad \lhd$$
$$\textbf{od}$$

This finishes the tactic application. The output of its application is

$$q, r := 0, a;$$
$$\textbf{do}\ \ r \ge b \rightarrow$$
$$q, r : \left[\left\{ \begin{matrix} b > 0 \land a = q * b + r \\ 0 \le r \land r \ge b \\ b > 0 \land a = q * b + r \\ 0 \le r \land 0 \le r < r_0 \end{matrix} \right\}, \right] \qquad \lhd$$
$$\textbf{od}$$

To finish the development we introduce the assignments in the body of the iteration. At each step of the iteration we must increment q by one and decrement

r by b.

$\sqsubseteq assignIV(q, r := q + 1, r - b)$
$\quad q, r := q + 1, r - b$

This law generates the following proof obligation.

$$(q = q_0) \land (r = r_0) \land b > 0 \land a = q * b + r \land 0 \leq r \land r \geq b \Rightarrow$$
$$b > 0 \land a = (q + 1) * b + (r - b) \land 0 \leq (r - b) \land 0 \leq (r - b) < r_0).$$

So, by applying the following tactic to the original specification

$\quad$ **law** $\ strPost((a = q * b + r \land 0 \leq r) \land \neg\ r \geq b);$
$\quad$ **tactic** $\ takeConjAsInv(b > 0, (q, r := 0, a), r);$
$\quad$ (**law** $\ skipInt\ \boxed{;}$
$\quad$ ($\boxed{\textbf{do}}$ **law** $assignIV(q, r := q + 1, r - b)$ $\boxed{\textbf{od}}$))

we get the following program

$\quad q, r := 0, a;$
$\quad$ **do** $\ r \geq b \rightarrow$
$\quad\quad\quad q, r := q + 1, r - b$
$\quad$ **od**

which correct implements the initial specification.

2.2.4 ReplConsByVar

Another common way of choosing an iteration invariant is replacing a constant in the specification postcondition by a variable. This strategy is captured by the tactic *replConsByVar*.

The tactic *replConsByVar* has five arguments: the declaration $newV : T$ of the fresh loop-index variable; the constant *cons* to be replaced; an invariant *invBound* on the loop indexes; the loop initialization $ivar := ival$; and an integer-valued

variant.

    Tactic *replConsByVar* (*newV* : *T*, *cons*, *invBound*, (*ivar* := *ival*), *variant*)
        applies to *w* : [*pre*, *post*] **do**

            law *varInt*(*newV* : *T*);
            $\boxed{\textbf{var}}$ **law** *strPost*($post[cons \setminus newV] \wedge \neg (newV \neq cons)$);
                tactic *takeConjAsInv*(*invBound*, (*ivar* := *ival*), *variant*); $\boxed{]\!]}$

        proof obligations

            $post[cons \setminus newV] \wedge \neg (newV \neq cons) \Rightarrow post$,
            $(invBound \wedge post[cons \setminus newV] \wedge \neg (newV \neq cons)) \Rightarrow$
                $post[cons \setminus newV] \wedge \neg (newV \neq cons)$,
            $pre \Rightarrow (invBound \wedge post[cons \setminus newV] \wedge invBound)[ivar \setminus ival]$

        program generated

            $[\![$ **var** *newV* : *T* •

                ivar := *ival*;
                do $newV \neq cons \rightarrow$

$$newV, w : \left[\left(\begin{array}{l} invBound \wedge post[cons \setminus newV] \\ newV \neq cons \\ invBound \wedge post[cons \setminus newV] \\ 0 \leq variant < variant_0 \end{array} \right), \right]$$

                od $]\!]$

      end

This tactic first introduces the new variable, then it strengthens the postcondition to replace the constant by the new variable. Afterwards it calls the tactic *takeConjAsInv* to introduce the iteration. The first proof obligation is generated by the law *strPost*, and the others are generated by the tactic *takeConjAsInv*. The first and the second proof obligations always hold.

Example. The following example presents a program to calculate, given two non-negative numbers a and b, the exponentiation a^b. Its formal specification is

$r : [a \geq 0 \wedge b \geq 0, r = a^b]$

The first step of the refinement is to apply the tactic *replConsByVar* with the loop index $x : \mathbb{Z}$, the constant to be replaced b, the invariant of the loop indexes $0 \leq x \leq b$, the variables initialization $(x, r := 0, 1)$, and the variant $b - x$ of the

iteration as arguments. First, the tactic introduces the new variable x which is used in the iteration. Then, the tactic strengthens the postcondition in order to make a relation between the new variable x and the constant b to be replaced.

$\sqsubseteq replConsByVar(x : \mathbb{Z}, b, 0 \leq x \leq b, (x, r := 0, 1), b - x)$
$\quad \sqsubseteq varInt(x : \mathbb{Z})$
$\quad\quad [\![\textbf{var} \ x : \mathbb{Z} \ \bullet$
$\quad\quad\quad r, x : [a \geq 0 \wedge b \geq 0, r = a^b] \qquad\qquad\qquad\qquad \triangleleft$
$\quad\quad]\!]$
$\quad \sqsubseteq strPost(r = a^x \wedge \neg \ x \neq b)$
$\quad\quad r, x : [a \geq 0 \wedge b \geq 0, r = a^x \wedge \neg \ x \neq b] \qquad\qquad \triangleleft$

This law generates the following proof obligation.

$$(r = a^x \wedge \neg \ x \neq b) \Rightarrow (r = a^b)$$

Then, the tactic calls the tactic $takeConjAsInv$, which first strengthens the postcondition in order to introduce the bound limits of the invariant.

$\quad \sqsubseteq takeConjAsInv(0 \leq x \leq b, (x, r := 0, 1), b - x)$
$\quad\quad \sqsubseteq strPost(0 \leq x \leq b \wedge r = a^x \wedge \neg \ x \neq b)$
$\quad\quad\quad r, x : [a \geq 0 \wedge b \geq 0, 0 \leq x \leq b \wedge r = a^x \wedge \neg \ x \neq b] \qquad \triangleleft$

This law generates the following proof obligation.

$$(0 \leq x \leq b \wedge r = a^x \wedge \neg \ x \neq b) \Rightarrow (r = a^x \wedge \neg \ x \neq b)$$

Then, the tactic $takeConjAsInv$ introduces a sequential composition. The first part is refined to the initialization of the iteration.

$\quad\quad \sqsubseteq seqComp(0 \leq x \leq b \wedge r = a^x)$
$\quad\quad\quad r, x : [a \geq 0 \wedge b \geq 0, 0 \leq x \leq b \wedge r = a^x] \qquad\qquad\qquad \triangleleft$
$\quad\quad\quad r, x : [0 \leq x \leq b \wedge r = a^x, 0 \leq x \leq b \wedge r = a^x \wedge \neg \ x \neq b] \qquad \text{(i)}$
$\quad\quad \sqsubseteq assign(x, r := 0, 1)$
$\quad\quad\quad x, r := 0, 1;$

This law generates the following proof obligation.

$$(a \geq 0 \wedge b \geq 0) \Rightarrow (0 \leq 0 \leq b \wedge 1 = a^0)$$

Finally, the second part of the sequential composition is refined by the tactic

$takeConjAsInv$ to an iteration.

$$(i) \sqsubseteq iter(\langle x \neq b \rangle, x - b)$$
$$\mathbf{do}\ x \neq b \rightarrow$$
$$r, x : \left[\left\{ \begin{array}{l} 0 \leq x \leq b \\ r = a^x \wedge x \neq b \end{array} \right), \\ \left(\begin{array}{l} 0 \leq x \leq b \\ r = a^x \wedge 0 \leq b - x \leq b - x_0 \end{array} \right) \right] \qquad \vartriangleleft$$
$$\mathbf{od}$$

To finish this development we should introduce the assignment in the body of the iteration as seen below.

$$\sqsubseteq assigIV(r, x := r * a, x + 1)$$
$$r, x := r * a, x + 1;$$

This law generates the following proof obligation.

$$(r = r_0 \wedge x = x_0 \wedge 0 \leq x \leq b \wedge r = a^x \wedge x \neq b) \Rightarrow$$
$$(0 \leq x + 1 \leq b \wedge r * a = a^{x+1} \wedge 0 \leq b - (x + 1) \leq b - x_0)$$

So, by applying the following tactic to the original specification.

$$\mathbf{tactic}\ replConsByVar(x : \mathbb{Z}, b, 0 \leq x \leq b, (x, r := 0, 1), b - x);$$
$$\boxed{\mathbf{var}}\ \mathbf{skip}\ \boxed{;}\ \boxed{\mathbf{do}}\ \mathbf{law}\ assignIV(r, x := r * a, x + 1)\ \boxed{\mathbf{od}}\ \boxed{]\!]}$$

We get the following program.

$$[\![\ \mathbf{var}\ \ x : \mathbb{Z} \bullet$$
$$x, r := 0, 1;$$
$$\mathbf{do}\ \ x \neq b \rightarrow r, x := r * a, x + 1\ \mathbf{od}\ ]\!]$$

This program uses a variable x, which is initialized with 0, in the iteration to count the number of times r, which is initialized with 1, is multiplied by a. In the end of the loop we have that $r = a^b$.

2.2.5   StrengInv

Sometimes we cannot simply replace a constant by a variable in the postcondition of the specification to determine the invariant. First, we have to strengthen the postcondition.

The tactic $strengInv$ has seven arguments: a variable declaration $newV_1 : T_1$, a predicate $streng$ and the rest of the arguments are the same as those taken by the

tactic *replConsByVar*. This tactic applies to a specification statement $w : [pre, post]$ to introduce an initialized iteration with $(post \wedge streng)[cons \setminus newV_2] \wedge invBound$ as invariant and *variant* as variant. The initialization is $ivar := ival$. In this tactic the predicate *invBound* states the range limits (typically 0 and *cons*) of the new variable $newV_2$ which is used as an indexing variable of the iteration. The new variable $newV_1$ is used as an auxiliary variable, and the predicate *streng* is used to make a link between the new variable introduced and the data used in the specification.

Tactic *strengInv*
  ($newV_1 : T_1, streng, newV_2 : T_2, cons, invBound, (ivar := ival), variant$)
  applies to $w : [\, pre, post \,]$ **do**

    law $varInt(\, newV_1 : T_1\,)$;
    $\boxed{\textbf{var}}$ **law** $strPost(\, post \wedge streng\,)$;
      tactic   *replConsByVar*
        ($newV_2 : T_2, cons, invBound, (ivar := ival), variant$) $\rbrack\!\rbrack$

  proof obligations

    $(post \wedge streng) \Rightarrow post$,
    $(post \wedge streng)[cons \setminus newV_2] \wedge \neg (newV_2 \neq cons) \Rightarrow (post \wedge streng)$,
    $(invBound \wedge (post \wedge streng)[cons \setminus newV_2] \wedge not(newV_2 \neq cons)) \Rightarrow$
      $(post \wedge streng)[cons \setminus newV_2] \wedge \neg (newV_2 \neq cons)$,
    $pre \Rightarrow (\, invBound \wedge (post \wedge streng)[cons \setminus newV_2] \wedge$
      $invBound\,)[ivar \setminus ival]$

  program generated

    $[\![\ \textbf{var}\ \ newV_1 : T_1 \bullet \textbf{var}\ newV_2 : T_2 \bullet$

      $ivar := ival$;
      do $newV_2 \neq cons \rightarrow$

$$
\begin{matrix}
newV_1, \\
newV_2, \\
w
\end{matrix}
\ :\
\begin{bmatrix}
\begin{pmatrix}
invBound \\
(post \wedge streng)[cons \setminus newV_2] \\
newV_2 \neq cons
\end{pmatrix}, \\
\begin{pmatrix}
invBound \\
(post \wedge streng)[cons \setminus newV_2] \\
0 \leq variant < variant_0
\end{pmatrix}
\end{bmatrix}
$$

      od $]\!]\!]$
  end

The first step of this tactic is to introduce a new variable which is used to strengthen

the postcondition. Then, this tactic strengthens the postcondition and calls the tactic *replConsByVar*.

Example. The following example derives a program which computes, given an array $f[0..n]$, the number of pairs (i, j) for which

$$0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0$$

The formal specification of this program is given below.

$$r : [n \geq 0, r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\}]$$

The principle of the program is to use a variable m to access each element of the array f. So, the range of m is $0 \leq m < n$. We use the variable s to count the number of non-positive elements in f, and the variable r to count the number of non-negative elements which are found after a non-positive element. The variables m, s, and r are initialized with 0. We use the tactic *strengInv* as seen below.

$\sqsubseteq$ *strengInv*
$\qquad (s : \mathbb{Z}, s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\},$
$\qquad m : \mathbb{Z}, n, 0 \leq m < n, (m, r, s := 0, 0, 0), n - m)$
$\qquad \sqsubseteq$ *varInt*$(s : \mathbb{Z})$
$\qquad\qquad [\![\textbf{ var } s : \mathbb{Z} \bullet$
$\qquad\qquad r, s : \left[\begin{array}{l} n \geq 0 \text{ ,} \\ \left(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \right) \end{array} \right] \quad \triangleleft$
$\qquad\qquad]\!]$
$\qquad \sqsubseteq$ *strPost*$(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\}$
$\qquad\qquad\qquad \wedge\ s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\})$
$\qquad r, s : \left[\begin{array}{l} n \geq 0 \text{ ,} \\ \left(\begin{array}{l} r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\} \end{array} \right) \end{array} \right] \quad \triangleleft$

This law generates the following proof obligation.

$$\begin{array}{l} (r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\ \quad s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\}) \Rightarrow \\ \quad\quad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \end{array}$$

Then, the tactic *strengInv* calls the tactic *replConsByVar* in order to introduce

the initialized iteration. The index of this iteration is the new variable m.

$\sqsubseteq$ $replConsByVar(m : \mathbb{Z}, n, 0 \leq m < n, (m, r, s), \langle 0, 0, 0 \rangle), n - m)$

$\sqsubseteq$ $varInt(m : \mathbb{Z})$

$\llbracket$ **var** $m : \mathbb{Z} \bullet$

$$
r, s, m : \left[\begin{array}{l} n \geq 0 \ , \\ \left(\begin{array}{l} r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < n \land f[i] \leq 0\} \end{array} \right) \end{array} \right] \quad \lhd
$$

$\rrbracket$

$\sqsubseteq$ $strPost(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \land f[i] \leq 0 \land f[j] \geq 0\} \land$
$\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < n \land f[i] \leq 0\})[n \setminus m] \land \neg\ m \neq n)$

$$
r, s, m : \left[\begin{array}{l} n \geq 0 \ , \\ \left(\begin{array}{l} r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \\ \neg\ m \neq n \end{array} \right) \end{array} \right] \quad \lhd
$$

This law generates the following proof obligation.

$$
(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \land f[i] \leq 0 \land f[j] \geq 0\} \land
$$
$$
s = \#\{i : \mathbb{N} \mid 0 \leq i < n \land f[i] \leq 0\})[n \setminus m] \land \neg\ m \neq n) \Rightarrow
$$
$$
(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \land f[i] \leq 0 \land f[j] \geq 0\} \land
$$
$$
s = \#\{i : \mathbb{N} \mid 0 \leq i < n \land f[i] \leq 0\}))
$$

The tactic $replConsByVar$ introduces the iteration using the tactic $takeConjAsInv$, as follows.

$\sqsubseteq$ $takeConjAsInv(0 \leq m \leq n, (m, r, s := 0, 0, 0), n - m)$

$\sqsubseteq$ $strPost(0 \leq m \leq n \land$
$\qquad\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \land$
$\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land \neg\ m \neq n)$

$$
r, s, m : \left[\begin{array}{l} n \geq 0 \ , \\ \left(\begin{array}{l} 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \\ \neg\ m \neq n \end{array} \right) \end{array} \right] \quad \lhd
$$

This law generates the following proof obligation.

$$
(0 \leq m \leq n \land r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \land
$$
$$
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land \neg\ m \neq n) \Rightarrow
$$
$$
(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \land
$$
$$
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land \neg\ m \neq n)
$$

First, the tactic $takeConjAsInv$ introduces a sequential composition and the

initialization of the iteration.

$$
\begin{aligned}
&\sqsubseteq\ seqComp(0 \leq m \leq n \ \wedge \\
&\qquad\qquad\quad r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \ \wedge \\
&\qquad\qquad\quad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\})
\end{aligned}
$$

$$
r, s, m : \left[\begin{array}{l}
n \geq 0 \ , \\
\left(\begin{array}{l}
0 \leq m \leq n \\
r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\}
\end{array}\right)
\end{array}\right] ; \ \vartriangleleft
$$

$$
r, s, m : \left[\begin{array}{l}
\left(\begin{array}{l}
0 \leq m \leq n \\
r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\}
\end{array}\right), \\
\left(\begin{array}{l}
0 \leq m \leq n \\
r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\
\neg\ m \neq n
\end{array}\right)
\end{array}\right] \quad (i)
$$

$$
\sqsubseteq\ assign(m, r, s := 0, 0, 0)
$$
$$
m, s, r := 0, 0, 0
$$

This law generates the following proof obligation.

$$
\begin{aligned}
n \geq 0 \Rightarrow & \\
& (0 \leq 0 \leq n \wedge 0 = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < 0 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
& \wedge\ 0 = \#\{i : \mathbb{N} \mid 0 \leq i < 0 \wedge f[i] \leq 0\})
\end{aligned}
$$

Finally, the tactic *takeConjAsInv* introduces the iteration.

$$
(i) \sqsubseteq\ iter(\langle m \neq n \rangle, n - m)
$$
$$
\textbf{do}\ \ m \neq n \to
$$

$$
r, s, m : \left[\begin{array}{l}
\left(\begin{array}{l}
0 \leq m \leq n \\
r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\
m \neq n
\end{array}\right), \\
\left(\begin{array}{l}
0 \leq m \leq n \\
r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\
s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\
0 \leq n - m < n - m_0
\end{array}\right)
\end{array}\right] \quad \vartriangleleft
$$

$$
\textbf{od}
$$

We should now refine the body of the iteration in order to get the resulting program. First, we introduce the increment of the iteration index m to the end of

each iteration, and then, we split the specification into two parts.

$\sqsubseteq \; fassign(m := m + 1)$

$$r, s, m : \left[\begin{pmatrix} 0 \le m \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m \land f[i] \le 0\} \\ m \ne n \end{pmatrix}, \begin{pmatrix} 0 \le m + 1 \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m + 1 \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m + 1 \land f[i] \le 0\} \\ 0 \le n - (m + 1) < n - m_0 \end{pmatrix} \right]; \quad \lhd$$

$m := m + 1$

$\sqsubseteq \; seqComp(0 \le m \le n \land$
$\qquad\qquad r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m + 1 \land f[i] \le 0 \land f[j] \ge 0\} \land$
$\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \le i < m \land f[i] \le 0\} \land m \ne n)$

$$r, s, m : \left[\begin{pmatrix} 0 \le m \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m \land f[i] \le 0\} \\ m \ne n \end{pmatrix}, \begin{pmatrix} 0 \le m \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m + 1 \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m \land f[i] \le 0\} \\ m \ne n \end{pmatrix} \right]; \quad \lhd$$

$$r, s, m : \left[\begin{pmatrix} 0 \le m \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m + 1 \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m \land f[i] \le 0\} \\ m \ne n \end{pmatrix}, \begin{pmatrix} 0 \le m + 1 \le n \\ r = \#\{i, j : \mathbb{N} \mid 0 \le i < j < m + 1 \land f[i] \le 0 \land f[j] \ge 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \le i < m + 1 \land f[i] \le 0\} \\ 0 \le n - (m + 1) < n - m_0 \end{pmatrix} \right] \quad (ii)$$

The first program of the composition is refined to an alternation which counts the number of pairs that satisfies the conditions seen in the description of this

example.

$\sqsubseteq \; alt(\langle f[m] < 0, f[m] \geq 0 \rangle)$
$\quad$ **if** $f[m] < 0 \rightarrow$

$$r, s, m : \left[\begin{array}{c} \left(\begin{array}{l} f[m] < 0 \\ 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\ m \neq n \end{array} \right), \\ \left(\begin{array}{l} 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\ m \neq n \end{array} \right) \end{array} \right] \quad \triangleleft$$

$\quad [\!] \; f[m] \geq 0 \rightarrow$

$$r, s, m : \left[\begin{array}{c} \left(\begin{array}{l} f[m] \geq 0 \\ 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\ m \neq n \end{array} \right), \\ \left(\begin{array}{l} 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \\ m \neq n \end{array} \right) \end{array} \right] \quad (iii)$$

$\quad$ **fi**

This law generates the following proof obligation.

$$(0 \leq m \leq n \wedge$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge m \neq n) \Rightarrow$$
$$(f[m] < 0 \vee f[m] \geq 0)$$

If $f[m] < 0$ we must skip.

$\sqsubseteq \; skipIntro()$
$\quad$ **skip**

This law generates the following proof obligation.

$$(f[m] < 0 \wedge 0 \leq m \leq n \wedge$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge m \neq n) \Rightarrow$$
$$(0 \leq m \leq n \wedge r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge m \neq n)$$

If $f[m] \geq 0$ we must increment the number of pairs r by s, the number of

non-positive number already found.

(iii) $\sqsubseteq$ $assign(r := r + s)$
    $r := r + s;$

This law generates the following proof obligation.

$$(f[m] \geq 0 \land 0 \leq m \leq n \land$$
$$\quad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\quad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land m \neq n) \Rightarrow$$
$$(0 \leq m \leq n \land$$
$$\quad r + s = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\quad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land m \neq n)$$

The second program of the sequential composition is refined to an alternation that counts the number of non-positive elements.

(ii) $\sqsubseteq$ $alt(\langle f[m] > 0, f[m] \leq 0 \rangle)$
    if $f[m] > 0 \rightarrow$

$$r, s, m : \left[\begin{array}{c} \left(\begin{array}{l} f[m] > 0 \\ 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \\ m \neq n \end{array} \right), \\ \left(\begin{array}{l} 0 \leq m + 1 \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m + 1 \land f[i] \leq 0\} \\ 0 \leq n - (m + 1) < n - m_0 \end{array} \right) \end{array} \right]$$ ◁

    $[\!] f[m] \leq 0 \rightarrow$

$$r, s, m : \left[\begin{array}{c} \left(\begin{array}{l} f[m] \geq 0 \\ 0 \leq m \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \\ m \neq n \end{array} \right), \\ \left(\begin{array}{l} 0 \leq m + 1 \leq n \\ r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \\ s = \#\{i : \mathbb{N} \mid 0 \leq i < m + 1 \land f[i] \leq 0\} \\ 0 \leq n - (m + 1) < n - m_0 \end{array} \right) \end{array} \right]$$ (iv)

    fi

This application of law *alt* generates the proof obligation which can be seen

below.

$$(0 \leq m \leq n \land$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land m \neq n) \Rightarrow$$
$$(f[m] < 0 \lor f[m] \geq 0)$$

If $f[m] > 0$ we must skip.

$$\sqsubseteq \; skipIntroIV()$$
$$\qquad \textbf{skip}$$

This law generates the following proof obligation.

$$(m = m_0 \land f[m] > 0 \land 0 \leq m \leq n \land$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land m \neq n) \Rightarrow$$
$$0 \leq m + 1 \leq n \land$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m + 1 \land f[i] \leq 0\} \land 0 \leq n - (m + 1) < n - m_0)$$

If $f[m] \leq 0$ we must increment the number of non-positive numbers s by one.

$$(iv) \sqsubseteq \; assignIV(s := s + 1)$$
$$\qquad s := s + 1$$

This law generates the following proof obligation.

$$(m = m_0 \land f[m] \leq 0 \land 0 \leq m \leq n \land$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \land f[i] \leq 0\} \land m \neq n) \Rightarrow$$
$$(0 \leq m + 1 \leq n \land$$
$$\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \land f[i] \leq 0 \land f[j] \geq 0\} \land$$
$$\qquad s + 1 = \#\{i : \mathbb{N} \mid 0 \leq i < m + 1 \land f[i] \leq 0\} \land$$
$$\qquad\qquad 0 \leq n - (m + 1) < n - m_0)$$

So, in this example, we can see that by applying the following tactic to the

original specification.

tactic $strengInv(s : \mathbb{Z}, s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\},$
$\qquad\qquad m : \mathbb{Z}, n, 0 \leq m < n, (m, r, s := 0, 0, 0), n - m);$
$\boxed{\textbf{var}}$

$\qquad\boxed{\textbf{var}}$
$\qquad\quad\textbf{skip}\,\boxed{;}$
$\qquad\quad\boxed{\textbf{do}}$
$\qquad\quad\textbf{law}\quad fassign(m := m + 1);$
$\qquad\quad($
$\qquad\qquad\quad(\textbf{law}\quad seqComp$
$\qquad\qquad\qquad\quad(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\}\,\wedge$
$\qquad\qquad\qquad\quad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge 0 \leq m \leq n \wedge m \neq n);$
$\qquad\qquad\qquad\quad(\textbf{law}\quad alt(\langle f[m] < 0, f[m] \geq 0\rangle);$
$\qquad\qquad\qquad\qquad\qquad\boxed{\textbf{if}}\ \textbf{law}\quad skipIntro()\,\boxed{[]}\ \textbf{law}\quad assign(r := r + s)\,\boxed{\textbf{fi}})$
$\qquad\qquad\qquad\boxed{;}$
$\qquad\qquad\qquad(\textbf{law}\quad alt(\langle f[m] > 0, f[m] \leq 0\rangle);$
$\qquad\qquad\qquad\qquad\qquad\boxed{\textbf{if}}\ \textbf{law}\quad skipIntroIV()\,\boxed{[]}\ \textbf{law}\quad assignIV(s := s + 1)\,\boxed{\textbf{fi}}))$
$\qquad\qquad\boxed{;}$
$\qquad\qquad\textbf{skip}$
$\qquad\quad)$
$\qquad\quad\boxed{\textbf{od}}$
$\qquad\boxed{]\!]}$
$\boxed{]\!]}$

we get the following program.

$[\![\ \textbf{var}\ s : \mathbb{Z} \bullet [\![\ \textbf{var}\ m : \mathbb{Z} \bullet$
$\qquad\qquad m, r, s := 0, 0, 0;$
$\qquad\qquad \textbf{do}\ \ m \neq n \rightarrow$
$\qquad\qquad\qquad \textbf{if}\ \ f[m] < 0 \rightarrow \textbf{skip};$
$\qquad\qquad\qquad\qquad [\!]\ f[m] \geq 0 \rightarrow r := r + s$
$\qquad\qquad\qquad \textbf{fi};$
$\qquad\qquad\qquad \textbf{if}\ \ f[m] > 0 \rightarrow \textbf{skip};$
$\qquad\qquad\qquad\qquad [\!]\ f[m] \leq 0 \rightarrow s := s + 1$
$\qquad\qquad\qquad \textbf{fi};$
$\qquad\qquad\qquad m := m + 1$
$\qquad\qquad \textbf{od}\ \ ]\!]\ ]\!]$

This program introduces two variables s and m. The first one is used to count
the number of non-positive numbers and the second one is used as an index in the

iteration. The variables m, r, and s are initialized with 0. Finally, the program checks each element of the array and increments r by s, if the element is non-negative, and s by one if the element is non-positive.

2.2.6 TailInvariant

This strategy is used when we want to develop an algorithm involving an iteration whose invariant is based on a function defined using tail recursion.

This tactic applies to a specification statement $w : [pre, post]$ to introduce an initialized iteration whose invariant is $invConj \wedge invBound$ and variant is $variant$, and a final assignment $ivar_2 := ival_2$, which typically is an assignment of an element of a sequence identified by an index found in the iteration body, to a variable. The initialization of the iteration is $ivar_1 := ival_1$.

> **Tactic** $tailInvariant(\ newV_1 : T_1, newV_2 : T_2, invConj, guard, invBound,$
> $(ivar_1 := ival_1), variant, (ivar_2 := ival_2)\ )$
> **applies to** $w : [\ pre, post\ ]$ **do**
>
> > **law** $varInt(\ newV_1 : T_1;\ newV_2 : T_2\ );$
> > $\boxed{\textbf{var}}$ **law** $seqComp(\ invConj \wedge \neg\ guard\ );$
> > **tactic** $takeConjAsInv(\ invBound, (ivar_1 := ival_1), variant)$
> > $\boxed{;}$ **law** $assign(ivar_2 := ival_2)\ )\boxed{]}$
>
> **proof obligations**
>
> > $invBound \wedge invConj \wedge \neg\ guard \Rightarrow invConj \wedge \neg\ guard,$
> > $pre \Rightarrow (\ invBound \wedge invConj\ )[ivar_1 \setminus ival_1],$
> > $invConj \wedge \neg\ guard \Rightarrow post[ivar_2 \setminus ival_2]$
>
> **program generated**
>
> > $[\![$ **var** $\ newV_1 : T_1;\ newV_2 : T_2 \bullet$
> >
> > > $ivar_1 := ival_1;$
> > > **do** $guard \rightarrow$
> > > > $newV_1, newV_2, w : [invBound \wedge invConj \wedge guard,$
> > > > > $invBound \wedge invConj \wedge 0 \leq variant < variant_0]$
> > > **od**;
> > > $ivar_2 := ival_2\ ]\!]$
>
> **end**

This tactic has eight arguments: the first two arguments are variable declarations, which are used as indexing variables in the iteration. The next argument is a predicate $invConj$. It is used to make a conjunction with the next argument, $guard$, which is also a predicate and represents the guard of the iteration. The next

argument, *invBound*, states the range limits of the invariant of the iteration. The assignment $ivar_1 := ival_1$ is the initialization of the iteration. An integer expression *variant* is the next argument and represents the variant of the iteration. Finally, the final assignment $ivar_2 := ival_2$ is used after the end of the loop.

The tactic *tailInvariant* first introduces two variables (arguments $newV_1 : T_1$ and $newV_2 : T_2$). Afterwards, it splits the body of the variable block into a sequential composition of two other specifications. The first defines the initialized iteration and the second the final assignment. In sequence, the tactic *takeConjAsInv* is applied to the first program using *invBound*, ($ivar_1 := ival_1$) and *variant* as arguments. The tactic also applies the law *assign* to the second program using the argument ($ivar_2 := ival_2$). The first and the second proof obligations are generated by the tactic *takeConjAsInv*. The third proof obligation is generated by the law *assign*.

Example. In this example, we develop a program which identifies the maximum element of an array. The function below has such behavior and is used in the development.

$$F.x.y \ A = \begin{cases} A.x & \text{if } x = y \\ F.(x+1).y & \text{if } A.x \le A.y \\ F.x.(y-1) & \text{if } A.x > A.y \end{cases}$$

It is a recursive function which, given two integers x and y, and an array A, returns the maximum element of the set $\{i : \mathbb{N} \mid x \le i \le y \bullet A.i\}$.

The specification of our program is

$$r : [n \ge 0, r = max\{i : \mathbb{N} \mid 0 \le i \le n \bullet A.i\}] \hspace{3cm} \triangleleft$$

We start our development by strengthening the post condition in order to use the function definition presented above.

$\sqsubseteq \ \ strPost(r = F.0.n)$
$\ \ \ \ \ \ r = [n \ge 0, r = F.0.n] \hspace{5cm} \triangleleft$

This law generates the following proof obligation

$$r = F.0.n \Rightarrow r = max\{i : \mathbb{N} \mid 0 \le i \le n \bullet A.i\}$$

Then, we use the tactic *tailInvariant*, which starts by introducing the variables

x and y used to calculate the function.

$\sqsubseteq$ $tailInvariant(x : \mathbb{N}, y : \mathbb{N}, F.x.y = F.0.n \wedge 0 \le x \le y \le n, x \ne y, true,$
$\qquad\qquad\qquad (x, y := 0, n), y - x)$
$\quad \sqsubseteq$ $varInt(x : \mathbb{N};\ y : \mathbb{N})$
$\qquad [\![\ \textbf{var}\ \ x : \mathbb{N};\ y : \mathbb{N} \bullet$
$\qquad\qquad y, x, r : [n \ge 0, r = F.0.n]$ $\qquad\qquad\qquad\qquad\qquad \lhd$
$\qquad]\!]$

The next step is to split the specification into two programs.

$\quad \sqsubseteq$ $seqComp(F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y)$
$\qquad y, x, r : [n \ge 0, F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y]$ $\qquad \lhd$
$\qquad y, x, r : [F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y, r = F.0.n]$ $\qquad (i)$

The first part of the program is refined to the initialized iteration using the tactic $takeConjAsInv$.

$\quad \sqsubseteq$ $takeConjAsInv(true, (x, y := 0, n), y - x)$
$\qquad \sqsubseteq$ $strPost(true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y)$
$\qquad\quad y, x, r : \left[\begin{array}{l} n \ge 0\ , \\ true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y \end{array} \right] \lhd$

This law generates the following proof obligation

$$(true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y) \Rightarrow$$
$$(F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y)$$

The tactic $takeConjAsInv$ splits the specification into two parts. The first part is refined to the initialization of the variables used in the iteration to which the second part is refined.

$\quad \sqsubseteq$ $seqComp(true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n)$
$\qquad y, x, r : [n \ge 0, true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n]$ $\qquad\qquad \lhd$
$\qquad y, x, r : \left[\begin{array}{l} true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n\ , \\ true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge \neg\, x \ne y \end{array} \right] (ii)$
$\quad \sqsubseteq$ $assign(x, y := 0, n)$
$\qquad x, y := 0, n;$

This law generates the following proof obligation

$$n \ge 0 \Rightarrow (true \wedge F.0.n = F.0.n \wedge 0 \le 0 \le n \le n)$$

Then, in order to introduce the iteration, we use the tactic $takeConjAsInv$ as

seen below.

$$(ii) \sqsubseteq \ iter(< x \neq i >, y - x)$$

$$\textbf{do} \ \ x \neq y \rightarrow$$

$$y, x, r : \left[\left(\begin{array}{l} true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \wedge x \neq y \end{array} \right), \left(\begin{array}{l} true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \\ 0 \leq y - x < y_0 - x_0 \end{array} \right) \right] \qquad (iii)$$

$$\textbf{od}$$

Finally, the tactic *tailInvariant* refines the second part of the program to the assignment $r := A.x$.

$$(i) \sqsubseteq \ assign(r := A.x)$$

$$r := A.x$$

This law generates the following proof obligation

$$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge \neg \, x \neq y) \Rightarrow A.x = F.0.n$$

At this point, we finish the application of tactic *tailInvariant*. Now, we refine the body of the iteration to an alternation as seen below.

$$(iii) \sqsubseteq \ alt(\langle A.x \leq A.y, A.x > A.y \rangle)$$

$$\textbf{if} \ \ A.x \leq A.y \rightarrow$$

$$y, x, r : \left[\left(\begin{array}{l} A.x \leq A.y \\ true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \\ x \neq y \end{array} \right), \left(\begin{array}{l} true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \\ 0 \leq y - x < y_0 - x_0 \end{array} \right) \right] \qquad \lhd$$

$$[\!] \ A.x > A.y \rightarrow$$

$$y, x, r : \left[\left(\begin{array}{l} A.x > A.y \\ true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \\ x \neq y \end{array} \right), \left(\begin{array}{l} true \wedge F.x.y = F.0.n \\ 0 \leq x \leq y \leq n \\ 0 \leq y - x < y_0 - x_0 \end{array} \right) \right] \qquad (iv)$$

$$\textbf{fi}$$

This law generates the following proof obligation

$$(true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge x \ne y) \Rightarrow$$
$$(A.x \le A.y \vee A.x > A.y)$$

If $A.x \le A.y$ we should increment the auxiliary variable x by one. Otherwise, we should decrement the auxiliary variable y by one. We refine the first guarded command.

$\sqsubseteq \ assignIV(x := x + 1)$
$\quad x := x + 1$

This law generates the following proof obligation

$$(A.x \le A.y \wedge true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge x \ne y) \Rightarrow$$
$$(true \wedge F.(x+1).y = F.0.n \wedge$$
$$0 \le x + 1 \le y \le n \wedge 0 \le y - (x+1) \le y_0 - x_0)$$

Now, we refine the second guarded command.

(iv) $\sqsubseteq \ assignIV(y := y - 1)$
$\quad y := y - 1$

This law generates the following proof obligation.

$$(A.x > A.y \wedge true \wedge F.x.y = F.0.n \wedge 0 \le x \le y \le n \wedge x \ne y) \Rightarrow$$
$$(true \wedge F.x.(y-1) = F.0.n \wedge 0 \le x \le y - 1 \le n \wedge$$
$$0 \le (y-1) - x \le y_0 - x_0)$$

We get the following program.

```
[ var x : ℕ; y : ℕ •
    x, y := 0, n;
    do x ≠ y →
        if A.x ≤ A.y → x := x + 1
         [] A.x > A.y → y := y - 1
        fi
    od
    r := A.x;
]
```

The strategies already presented do not have included procedure concepts. The next sections introduce tactics which work with procedures.

2.2.7 ProcNoArgs

Most of the laws in the refinement calculus are very basic. It is very common to use a set of them together. It is the case of the tactic *procNoArgs* which introduces a procedure block that declares a procedure with no parameters. It has two arguments: the name of the procedure which is introduced and its body. This tactic applies to a program p, introduces the procedure, and makes one or more calls to it, depending on the program p.

> **Tactic** *procNoArgs* (*procName*, *procBody*)
>   **applies to** p **do**
>
>   **law** *procNoArgsIntro*(*procName*, *procBody*);
>   **law** *procNoArgsCall*( )
>
>   **program generated**
>
> $[\![$ **proc** *procName* $\mathrel{\widehat{=}}$ *procBody* $\bullet$ $p[procBody \setminus procName]]\!]$
>   **end**

The program $p[procBody \setminus procName]$ is that obtained by replacing all occurencies of *procBody* in p with *procName*.

Example. As an example of the use of the tactic *procNoArgs* we present again the development of the second example of Section 1.2. This example, derives a program which, given three numbers p, q, and r, exchanges their values such that $p \leq q \leq r$. The specification of this program is

$$p, q, r : [p \leq q \leq r \wedge \lfloor p, q, r \rfloor = \lfloor p_0, q_0, r_0 \rfloor]$$

We have started the derivation by splitting the program in three assignments. The last assignment sorts p and q.

$\sqsubseteq$ $fassign(p, q := (p \sqcap q), (p \sqcup q))$
    $p, q, r : [(p \sqcap q) \leq (p \sqcup q) \leq r \wedge \lfloor (p \sqcap q), (p \sqcup q), r \rfloor = \lfloor p_0, q_0, r_0 \rfloor];$ $\lhd$
    $p, q := (p \sqcap q), (p \sqcup q)$

The second assignment sorts q and r, and finally, the first assignment also sorts

p and q.

$\sqsubseteq\ fassign(q, r := (q \sqcap r), (q \sqcup r))$

$$p, q, r : \left[\begin{array}{c} (p \sqcap (q \sqcap r)) \leq (p \sqcup (q \sqcap r)) \leq (q \sqcup r) \\ \lfloor (p \sqcap (q \sqcap r)), (p \sqcup (q \sqcap r)), (q \sqcup r) \rfloor = \lfloor p_0, q_0, r_0 \rfloor \end{array} \right] ; \qquad \triangleleft$$

$\qquad q, r := (q \sqcap r), (q \sqcup r)$

$\sqsubseteq\ assignIV(p, q := (p \sqcap q), (p \sqcup q))$

$\qquad p, q := (p \sqcap q), (p \sqcup q) \qquad\qquad\qquad\qquad\qquad\qquad\qquad \triangleleft$

This law generates the following proof obligation.

$$\begin{aligned} p = p_0 &\wedge q = q_0 \wedge r = r_0 \Rightarrow \\ &((p \sqcap q) \sqcap ((p \sqcup q) \sqcap r)) \leq ((p \sqcap q) \sqcup ((p \sqcup q) \sqcap r)) \leq ((p \sqcup q) \sqcup r) \\ &\lfloor ((p \sqcap q) \sqcap ((p \sqcup q) \sqcap r)), ((p \sqcap q) \sqcup ((p \sqcup q) \sqcap r)), ((p \sqcup q) \sqcup r) \rfloor = \\ &\lfloor p_0, q_0, r_0 \rfloor \end{aligned}$$

Now, we use the tactic *procNoArgs* to introduce the procedure and use it in the body of the program.

$\sqsubseteq\ procNoArgs(sort, (p, q := p \sqcap q, p \sqcup q))$

$\quad \sqsubseteq\ procNoArgsIntro(sort, (p, q := p \sqcap q, p \sqcup q))$

$\qquad [\![\ \textbf{proc}\ \ sort \mathrel{\widehat{=}} p, q := p \sqcap q, p \sqcup q\ \bullet$

$\qquad\qquad p, q := (p \sqcap q), (p \sqcup q);$

$\qquad\qquad q, r := (q \sqcap r), (q \sqcup r);$

$\qquad\qquad p, q := (p \sqcap q), (p \sqcup q)]\!]$

$\quad \sqsubseteq\ procNoArgsCall()$

$\qquad [\![\ \textbf{proc}\ \ sort \mathrel{\widehat{=}} p, q := p \sqcap q, p \sqcup q\ \bullet$

$\qquad\qquad sort;\ q, r := (q \sqcap r), (q \sqcup r);\ sort]\!]$

So, by applying the following tactic to the initial specification.

$\qquad \textbf{law}\ \ fassign(p, q := (p \sqcap q), (p \sqcup q));$
$\qquad (\textbf{law}\ \ fassign(q, r := (q \sqcap r), (q \sqcup r))\ \fbox{;}\, \textbf{skip});$
$\qquad (\textbf{law}\ \ assignIV(p, q := (p \sqcap q), (p \sqcup q))\ \fbox{;}\, \textbf{skip}\ \fbox{;}\, \textbf{skip});$
$\qquad \textbf{tactic}\ \ procNoArgsIntro(sort, (p, q := p \sqcap q, p \sqcup q));$
$\qquad \textbf{tactic}\ \ procNoArgsCall()$

we get the following program.

$\qquad [\![\ \textbf{proc}\ \ sort \mathrel{\widehat{=}} p, q := p \sqcap q, p \sqcup q\ \bullet$
$\qquad\qquad sort;\ q, r := (q \sqcap r), (q \sqcup r);\ sort]\!]$

This program first calls the procedure *sort* to sort p and q. Then, it sorts q and r. Finally, it calls again the procedure *sort* to sort p and q again.

2.2.8 ProcCalls

Before describing a tactic which introduces procedures with arguments, it is useful to define a tactic that introduces parameterized commands. The tactic *procCalls* takes as arguments two lists. The first is a list of parameter declarations and the second is a list of arguments. The declarations have the form *pasMech v : T* where *pasMech* defines how the arguments are passed, by value(**val**), by result(**res**) or by value-result(**val-res**), *v* is the name of the argument, and T its type.

This tactic tries to derive an application of a procedure call with the given parameters to the given arguments. With this purpose, it tries to apply each of the laws that introduces parameterized commands. If one of them succeeds, the tactic goes on with the tail of the list, else, the tactic behaves like **skip** and finishes.

> **Tactic** *procCalls* (*pars, args*)
>   **applies to** *w : [pre, post]* **do**
>
>   **law** *callByValue(head' args, head' pars)*;
> $\boxed{\textbf{val}}$ (**tactic** *procCalls(tail pars, tail args)*) $\mid$
>   **law** *callByValueIV(head' args, head' pars)*;
> $\boxed{\textbf{val}}$ (**law** *contractFrame(head' pars)*;
>   **tactic** *procCalls(tail pars, tail args)*) $\mid$
>   **law** *callByResult(head' args, head' pars)*;
> $\boxed{\textbf{res}}$ (**tactic** *procCalls(tail pars, tail args)*) $\mid$
>   **law** *callByValueResult(head' args, head' pars)*;
> $\boxed{\textbf{val-res}}$ (**tactic** *procCalls(tail pars, tail args)*) $\mid$
>   **skip**
>   **end**

The function *head'* applies to a list, and gives another list that contains just the head of the given list, or is empty if the given list is empty. The function *tail* returns the given list removing its first element. These functions *head'* and *tail* are not defined in ArcAngel's semantics and are not supported by Gabriel. However, implementing these functions in Gabriel is an interesting task that is left as future work.

This tactic is recursive and can generate applications of parameterized commands whose bodies can include further applications. An example of the use of this tactic is presented in the next section.

2.2.9 ProcArgs

Now we can define the tactic *procArgs* which introduces a parameterized procedure in the scope of the program and makes calls to this procedure. Here the argu-

ment *pars* is a parameter declaration. We make use of the function *seqToList* to convert *args*, a semicolon-separated sequence of argument declarations, to a list of declarations.

> **Tactic** *procArgs* (*procName*, *body*, *pars*, *args*)
>   **applies to** *p* **do**
>
>   **law** *progArgsIntro*(*procName*, *body*, *pars*);
> $\boxed{\textbf{pmain}}$
>
>   **tactic** *procCalls*(*seqToList pars*, *args*);
>   *exhaust*(**law** *multiArgs*())
>
> $\rrbracket$;
>   **law** *procArgsCall*()
> **end**

This tactic first introduces a procedure with arguments using the law *procArgsIntro*. Then the tactic uses *procCalls* to introduce applications of parameterized commands with parameters *pars* to arguments *args*. Finally, the tactic uses the law *multiArgs* to nested applications of parameterized commands. The definition of *exhaust* can be found in Section 2.3.4.

The function *seqToList* is also not defined in ArcAngel's semantics. However, implementing this function in Gabriel is another interesting task that is left as future work.

Example. We use the square root program, presented in Section 1.2, as an example. Its specification is

$$x : [0 \le x, x^2 = x_0]$$

Our development is simplified by the use of tactic *procArgs*, which starts with the introduction of the *sqrts* procedure.

$x : [0 \le x, x^2 = x_0]$
$\sqsubseteq$ $procArgs(sqrts, b : [0 \le a, b^2 = a], (\textbf{val} \ \ a : \mathbb{R}; \ \textbf{res} \ \ b : \mathbb{R}), \langle x, x \rangle)$
    $\sqsubseteq$ $procArgsIntro(sqrts, b : [0 \le a, b^2 = a], (\textbf{val} \ \ a : \mathbb{R}; \ \textbf{res} \ \ b : \mathbb{R}))$
        $\llbracket$ **proc** $sqrts \ \hat{=} \ (\textbf{val} \ \ a : \mathbb{R}; \ \textbf{res} \ \ b : \mathbb{R} \bullet b : [0 \le a, b^2 = a]) \bullet$
           $x : [0 \le x, x^2 = x_0]$ $\lhd$
        $\rrbracket$

Then, using the tactic *procCalls*, this tactic attempts to introduce the substitu-

tions of the variable x.

$$\sqsubseteq procCalls(\langle \mathbf{val} \quad a : \mathbb{R}, \mathbf{res} \quad b : \mathbb{R} \rangle, \langle x, x \rangle)$$
$$\quad \sqsubseteq callByValueIV(x, a)$$
$$\qquad (\mathbf{val} \quad a : \mathbb{R} \bullet x, a : [0 \le a, x^2 = a_0])(x) \qquad\qquad \triangleleft$$
$$\quad \sqsubseteq contractFrame(a)$$
$$\qquad (\mathbf{val} \quad a : \mathbb{R} \bullet x : [0 \le a, x^2 = a])(x) \qquad\qquad \triangleleft$$
$$\quad \sqsubseteq procCalls(\langle \mathbf{res} \quad b : \mathbb{R} \rangle, \langle x \rangle)$$
$$\qquad \sqsubseteq callByResult(x, b)$$
$$\qquad\quad (\mathbf{res} \quad b : \mathbb{R} \bullet b : [0 \le a, b^2 = a])(x) \qquad\qquad \triangleleft$$
$$\qquad \sqsubseteq procCalls(\langle \rangle, \langle \rangle)$$
$$\qquad\quad (\mathbf{res} \quad b : \mathbb{R} \bullet b : [0 \le a, b^2 = a])(x)$$

Then it uses the tactical *exhaust* to put our two kinds of substitution together.

$$\sqsubseteq exhaust(\mathbf{law} \ multiArgs())$$
$$\quad \sqsubseteq multiArgs()$$
$$\qquad (\mathbf{val} \quad a : \mathbb{R}, \mathbf{res} \quad b : \mathbb{R} \bullet b : [0 \le a, b^2 = a])(x, x) \qquad\qquad \triangleleft$$

Finally, the tactic introduces the procedure call.

$$\sqsubseteq procArgsCall()$$
$$\quad sqrts(x, x)$$

So, by applying the following tactic to the initial specification.

$$\mathbf{tactic} \ procArgs(sqrts, (\mathbf{val} \quad a : \mathbb{R}; \ \mathbf{res} \quad b : \mathbb{R}), b : [0 \le a, b^2 = a], \langle x, x \rangle)$$

we get the following program.

$$[\![\ \mathbf{proc} \quad sqrts \mathrel{\widehat{=}} (\mathbf{val} \quad a : \mathbb{R}, \mathbf{res} \quad b : \mathbb{R} \bullet b : [0 \le a, b^2 = a]) \bullet$$
$$\quad sqrts(x, x)$$
$$]\!]$$

This program uses the procedure *sqrts* to calculate the square root of a given number.

2.2.10 RecProcArgs

This tactic is useful when we want to develop a recursive procedure and have to

introduce a variant block with a parameterized procedure. Its definition is

> **Tactic** *recProcArgs*
>   (*procName*, *body*, *variantName*, *variantExp*, *args*, *varsProcF*)
>   **applies to** *p* **do**
>
>   **law** *variantIntro*(*procName*, *body*, *variantName*, *variantExp*, *args*);
> $\boxed{\textbf{pmainvariant}}$
>
>   **tactic** *procCalls*(*seqToList args*, *varsProcF*);
>   *exhaust*(**law** *multiArgs*())
> $\boxed{]\!]}$;
>   **law** *procArgsVariantBlockCall*();
> **end**

This tactic first introduces a variant block using the law *variantIntro*. Then, as in the previous tactic, it uses the law *procCalls*. Finally, it uses the tactic *exhaust*(**law** *multiArgs*()) to introduce an application of a parameterized command. Afterwards, the tactic uses the law *procArgsVariantBlockCall* to introduce a procedure call in the main program of the variant block.

Example.

We present the factorial program seen in Section 1.2 as an example. We have presented the following specification as the initial specification.

$$f : [f = n! * 1] \hspace{6cm} \triangleleft$$

We start by using the tactic *recProcArgs*. The variant introduction is the first step of this tactic.

$\sqsubseteq$ *recProcArgs*
    $(fact, V, m, (f : [f = m! * k]), \textbf{val}\ \ m, k : \mathbb{N}, \langle n, 1 \rangle)$
    $\sqsubseteq$ *variantIntro*$(fact, V, m, (f : [f = m! * k]), \textbf{val}\ \ m, k : \mathbb{N})$
        $[\![\ \textbf{proc}\ \ fact \mathrel{\widehat{=}} (\textbf{val}\ \ m, k : \mathbb{N} \bullet f : [V = m, f = m! * k])$
            $\textbf{variant}\ \ V\ \textbf{is}\ m \bullet$
            $f : [f = n! * 1] \hspace{4cm} \triangleleft$
        $]\!]$

Then, using the tactic *procCalls*, this tactic attempts to introduce a substitution

by value of the variables m and k.

$\sqsubseteq$ $procCalls(\langle m, k\rangle, \langle n, 1\rangle)$
    $\sqsubseteq$ $callByValue(m, k : \mathbb{N}, \langle n, 1\rangle)$
        $[\![$ **proc** $fact \mathrel{\widehat{=}} (\mathbf{val}\ \ m, k : \mathbb{N} \bullet f : [V = m, f = m! * k])$
            variant V **is** $m \bullet$
            $(\mathbf{val}\ \ m, k : \mathbb{N} \bullet f : [f = m! * k])(n, 1)$
        $]\!]$ $\triangleleft$

The next step is to introduce the procedure call in the variant block.

$\sqsubseteq$ $procArgsVariantBlockCall()$
    $[\![$ **proc** $fact \mathrel{\widehat{=}} (\mathbf{val}\ \ m, k : \mathbb{N} \bullet \{V = m\}\ f : [f = m! * k])$ $\triangleleft$
        variant V **is** $m \bullet$
        $fact(n, 1)$
    $]\!]$

This finishes the tactic application. Now, we start to develop the body of the procedure $fact$. First, we introduce an alternation in order to check if we must stop the recursion ($m = 0$) or not ($m > 0$).

$\sqsubseteq$ $alt(\langle m = 0, m > 0\rangle)$
    if $m = 0 \rightarrow f : [V = m \wedge m = 0, f = m! * k]$ $\triangleleft$
        $[\!]\ m > 0 \rightarrow f : [V = m \wedge m > 0, f = m! * k]$ (i)
    fi

This law application generates the following proof obligation.

$$V = m \Rightarrow m = 0 \vee m > 0$$

The base case is when we have m equals to 0. In such case, k must receive the value of the variable f.

$\sqsubseteq$ $assign(f := k)$
    $f := k$

The following proof obligation is generated in this case.

$$V = m \wedge m = 0 \Rightarrow k = 0! * k$$

If $m > 0$ we should make a recursive call. We first strengthen the postcondition

in order to make a recursive call of procedure *fact*.

(i) $\sqsubseteq$ *strPost*$(f = (m - 1)! * m * k)$
  $f : [V = m \wedge m > 0, f = (m - 1)! * m * k]$ ◁

  We have the following proof obligation

  $$f = (m - 1)! * m * k \Rightarrow f = m! * k$$

  Then we introduce a substitution by value and weaken the precondition.

$\sqsubseteq$ *callByValue*$(\langle m, k \rangle, \langle m - 1, m * k \rangle)$
  (**val** $m, k : \mathbb{N} \bullet f : [V = m + 1 \wedge m + 1 > 0, f = m! * k]$)
  $(m - 1, m * k)$ ◁
$\sqsubseteq$ *weakPre*$(0 \le m < V)$
  $f : [0 \le m < V, f = m! * k])$

  This law generates the following proof obligation

  $$V = m + 1 \wedge m + 1 > 0 \Rightarrow 0 \le m < V$$

  Now we have the following program.

  $[\![$ **proc** *fact* $\mathrel{\hat{=}}$ (**val** $m, k : \mathbb{N} \bullet$
     if $m = 0 \rightarrow f := k$
     $[\!]$ $m > 0 \rightarrow f : [0 \le m < V, f = m! * k]$
     fi)
    variant V **is** $m \bullet$
  $fact(n, 1)]\!]$

  Finally, we refine this program by introducing a recursive call to the procedure *fact* as follows.

$\sqsubseteq$ *recursiveCall*()
  $[\![$ **proc** *fact* $\mathrel{\hat{=}}$ (**val** $m, k : \mathbb{N} \bullet$
     if $m = 0 \rightarrow f := k$
     $[\!]$ $m > 0 \rightarrow fact(m - 1, m * k)$
     fi)
    variant V **is** $m \bullet$
  $fact(n, 1)]\!]$

  The last application generates the following proof obligation.

  $f : [V = n, f = m! * k]$
  $\sqsubseteq$
     if $m = 0 \rightarrow f := k$
     $[\!]$ $m > 0 \rightarrow f : [0 \le m < V, f = m! * k]$
  fi

  The program obtained calculates the factorial of a given number.

50

2.3 ArcAngel's Semantics

In this section we present ArcAngel's semantics which is based in Angel semantics. It, however, takes into account the particularities of the refinement calculus, in which a transformation rule consists of refinement laws. The application of a refinement law to a program and returns a new program, but also proof obligations.

Tactics are applied to a pair: the first element of this pair is a program to which the tactic is applied; the second element is the set of proof obligations generated to obtain this program. This pair is called *RCell* (*refinement cell*), and is defined as follows:

$$RCell == program \times \mathbb{P} \, predicate$$

The result of a tactic application is a possibly infinite list of *RCell*s that contains all possible outcomes of its application: every program it can generate, together with the corresponding proof obligations (existing obligations and those generated by the tactic application). Different possibilities arise from the use of alternation, and the list can be infinite, since the application of a tactic may run indefinitely. If the application of some tactic fails, then the empty list is returned.

$$Tactic == RCell \nrightarrow \text{pfiseq} \, RCell$$

The type pfiseq *RCell* is that of possibly infinite lists of *RCell*s. We use the model for infinite lists proposed in [19]. This is summarized in Appendix B.

In order to give semantics to named laws and tactics, we need to maintain two appropriate environments.

$$LEnv == name \nrightarrow \text{seq} \, argument \nrightarrow program \nrightarrow RCell$$

$$TEnv == name \nrightarrow \text{seq} \, argument \nrightarrow Tactic$$

A law environment records the set of known laws; it is a partial function whose domain is the set of the names of these laws. For a law environment Γ_L and a given law name n, we have that $\Gamma_L \, n$ is also a partial function: it relates all valid arguments of n to yet another function. For a valid argument a, we have that $\Gamma_L \, n \, a$ relates all the programs to which n can be applied when given arguments a; the result is a refinement cell.

For example, let Γ_L be an environment that records the law *strPost*. For a predicate $post'$ and a program $w : [\, pre, post \,]$ we have

$$\Gamma_L \, strPost \, post' \, (\, w : [\, pre, post \,]\,) = (\, w : [\, pre, post' \,], \{\, post' \Rightarrow post \,\}\,)$$

This means that if we apply *strPost* with argument $post'$ to $w : [\, pre, post \,]$, we change its postcondition to $post'$, but we must prove that $post' \Rightarrow post$. We are

51

interested in environments that record at least the laws of Morgan's refinement calculus in [23] and in Appendix C.

Similarly, a tactic environment is a function that takes a tactic name and a list of arguments, and returns a *Tactic*.

2.3.1 Tactics

We define the semantic function for tactics inductively; it has the type

$$[\![\, \_ \,]\!] : tactic \rightarrow LEnv \rightarrow TEnv \rightarrow Tactic$$

The basic tactic **law** $n(a)$ is that which applies a simple law to an *RCell*.

$[\![\, \textbf{law}\ n(a)\,]\!]\ \Gamma_L\ \Gamma_T\ (p, pobs) =$

> **if** $n \in \mathrm{dom}\,\Gamma_L \land a \in \mathrm{dom}(\Gamma_L\ n) \land p \in \mathrm{dom}(\Gamma_L\ n\ a)$
> **then**
> > **let** $(newp, npobs) = \Gamma_L\ n\ a\ p$ **in** $[(newp, pobs \cup npobs)]$
> **else** $[\,]$

We check if the law name n is in the law environment Γ_L, and if the arguments a and program p are appropriate. If these conditions hold, then the tactic succeeds, and returns a list with a new *RCell*. The program is transformed by applying the law to the program p; the new proof obligations are added to the proof obligations *pobs* of the original *RCell*. Otherwise, the tactic fails with the empty list as result. We use angle brackets to delimit finite lists; possibly infinite lists are delimited by square brackets.

In this work, we use a simple approach for expression arguments. For us, they are used as they were already evaluated. However, they should be evaluated before being used. This is left as future work.

The semantics of **tactic** $n(a)$ is similar to that of the **law** construct. Its definition is

$[\![\, \textbf{tactic}\ n(a)\,]\!]\ \Gamma_L\ \Gamma_T\ r =$

> **if** $n \in \mathrm{dom}\,\Gamma_T \land a \in \mathrm{dom}\,\Gamma_T\ n$
> **then** $\Gamma_T\ n\ a\ r$
> **else** $[\,]$

If the tactic is in the tactic environment, and if the arguments are valid, then the tactic succeeds; it returns the result of applying the tactic to the arguments. Otherwise, the tactic fails.

The tactic **skip** returns its argument unchanged; the tactic **fail** always fails.

$$[\![\,\textbf{skip}\,]\!]\,\Gamma_L\,\Gamma_T\,r = [\,r\,]$$

$$[\![\,\textbf{fail}\,]\!]\ \ \Gamma_L\,\Gamma_T\,r = [\,]$$

The sequence operator applies its first tactic to its argument, producing a list of cells; it then applies the second tactic to each member of this list; finally, this list-of-lists is flattened to produce the result.

$$[\![\,t_1;\ t_2\,]\!]\,\Gamma_L\,\Gamma_T = {}^{\infty}\!\!/ \cdot (\,[\![\,t_2\,]\!]\,\Gamma_L\,\Gamma_T\,)* \cdot (\,[\![\,t_1\,]\!]\,\Gamma_L\,\Gamma_T\,)$$

For a total function $f : A \rightarrow B$, $f* : \text{pfiseq}\,A \rightarrow \text{pfiseq}\,B$ is the map function that operates on a list by applying f to each of its elements; the operator $\cdot$ is used to compose functions; and ${}^{\infty}\!\!/$ is the distributed concatenation operation. Formal definitions of these operators and others to follow can be found in Appendix B.

The semantics of the alternation operator is given by concatenation: the possible outcomes of each individual tactic are joined to give the list of possible outcomes of the alternation.

$$[\![\,t_1 \mid t_2\,]\!]\,\Gamma_L\,\Gamma_T = {}^{\infty}\!\!/ \cdot [\,(\,[\![\,t_1\,]\!]\,\Gamma_L\,\Gamma_T\,),(\,[\![\,t_2\,]\!]\,\Gamma_L\,\Gamma_T\,)\,]^{\circ}$$

The function $^{\circ}$ applies a list of functions to a single argument and returns a list containing the results of the applications.

The cut operator applies its tactic to its argument, taking the first result (if it exists) and discarding the rest; if there is no first result, then the cut tactic fails.

$$[\![\,!\,t\,]\!]\,\Gamma_L\,\Gamma_T = head' \cdot (\,[\![\,t\,]\!]\,\Gamma_L\,\Gamma_T\,)$$

The recursion operator μ has a standard definition [9] as a least fixed point. For a continuous function f from tactics to tactics, we have that

$$(\,\mu\,X \bullet f(X)\,) = \bigsqcup\{\,i : \mathbb{N} \bullet f^i(\textbf{abort})\,\}$$

where f^i represents i applications of f.

This definition makes sense if the set of tactics is a complete lattice. First, we define a partial order for lists: the completely-undefined list, denoted $\bot$, is the least-defined element in the set. It is a partial list, as is any list that ends with $\bot$. One list is less than another, $s_1 \sqsubseteq_{\infty} s_2$, whenever they are equal, or the first is a partial list that forms an initial subsequence of the second. In the model we adopt, an infinite list is a limit of a directed set of partial lists.

For tactics, as they are partial functions, we may lift the ordering on lists of

*RCell*s to an ordering on tactics. We can say that

$$t_1 \sqsubseteq_T t_2 \Leftrightarrow (\forall\, r : RCell \bullet t_1\, r \sqsubseteq_\infty t_2\, r\,)$$

As the set of lists of *RCell*s is a complete lattice, the set of tactics is also a complete lattice, using the order above [9].

The bottom element is used in the semantics of **abort**, which runs indefinitely.

$$[\![\mathbf{abort}]\!]\, \Gamma_L\, \Gamma_T = \bot$$

The tactics **succs** t and **fails** t are defined as follows

$$\mathbf{succs}\, t\, r = (\mathbf{if}\; t\, r = \bot \;\mathbf{then\; abort\; else}\; (\mathbf{if}\; t\, r = [\,]\; \mathbf{then\; fail\; else\; skip})\, r)$$

$$\mathbf{fails}\, t\, r = (\mathbf{if}\; t\, r = \bot \;\mathbf{then\; abort\; else}\; (\mathbf{if}\; t\, r = [\,]\; \mathbf{then\; skip\; else\; fail})\, r)$$

If the application of t aborts, so do **succs** t and **fails** t. If it fails, then **succs** t fails and **fails** t skips. Finally, if it succeeds, **succs** t skips and **fails** t fails.

2.3.2 Structural Combinators

The structural combinators apply tactics to components of a program independently (and so can be thought of as in parallel), and then reassemble the results in all possible ways. There is one combinator for each construct in the programming language.

As already explained, the structural combinator $t_1 \,\fbox{;}\, t_2$ applies to a sequential composition $p_1;\ p_2$. Independently, t_1 is applied to p_1 and t_2 is applied to p_2; the resulting alternatives are assembled into pairs, so that the first is an alternative outcome from t_1 and the second is an alternative outcome from t_2; finally, these pairs are combined with the sequential composition tactical.

$$[\![\, t_1 \,\fbox{;}\, t_2 \,]\!]\, \Gamma_L\, \Gamma_T\, (p_1;\ p_2, pobs) =$$
$$\Omega_{;}*\,(\,\Pi(\,\langle\,(\,([\![\, t_1 \,]\!]\, \Gamma_L\, \Gamma_T\,)\,(p_1, pobs),\, ([\![\, t_2 \,]\!]\, \Gamma_L\, \Gamma_T\,)\,(p_2, pobs)\,)\,\rangle\,)\,)$$

The distributed cartesian product operator Π is defined in Appendix B. The sequential combination function $\Omega_{;}$ sequentially composes the programs and unites the proof obligations. The function $\Omega_{;}$ is a partial function because it is defined only for lists with two elements.

$$\Omega_{;} : RCell* \nrightarrow RCell$$

$$\Omega_{;}\, \langle(p_1, pobs_1), (p_2, pobs_2)\rangle = (p_1;\ p_2, pobs_1 \cup pobs_2)$$

The semantics of the structural combinators $\fbox{if}\,\_\,\fbox{fi}$ and $\fbox{do}\,\_\,\fbox{od}$ use a few functions

which are explained and defined as they are needed; we start by giving a simple example. Consider the *RCell*

$$(\textbf{if } x \geq 0 \rightarrow x : [\, x \geq 0, x > y \,] \; [] \; x < 0 \rightarrow x : [\, x < 0, x < z \,] \; \textbf{fi}, \{x \geq 1\})$$

and the tactic

$$\boxed{\textbf{if}} \; \textbf{law } \textit{weakPre}(\textit{true}) \,|\, \textbf{law } \textit{assign}(\, x := x + 1\,) \boxed{[]} \; \textbf{law } \textit{assign}(\, x := x - 1\,) \boxed{\textbf{fi}}$$

First of all, we extract the commands from each branch of the conditional; the function *extractP* does this for us.

$$\textit{extractP} :: \textit{GuardedCommand} \rightarrow \text{pfseq } \textit{program}$$
$$\textit{extractP } G \rightarrow P = \langle P \rangle$$
$$\textit{extractP } G \rightarrow P \; [] \; GC = P : (\textit{extractP } GC)$$

In our example we have

$$\textit{extractP}(\, x \geq 0 \rightarrow x : [\, x \geq 0, x > y \,] \; [] \; x < 0 \rightarrow x : [\, x < 0, x < z \,] \,)$$
$$= \langle\, x : [\, x \geq 0, x > y \,], x : [\, x < 0, x < z \,] \,\rangle$$

The function Φ_r constructs an *RCell* with a given program p and an empty set of proof obligations. We map Φ_r over our list of commands.

$$\Phi_r :: \textit{program} \rightarrow \textit{RCell}$$
$$\Phi_r \; p = (p, \emptyset)$$

In our example we have

$$\Phi_r * (\, \langle x : [\, x \geq 0, x > y \,], x : [\, x < 0, x < z \,] \rangle\,) =$$
$$\langle(\, x : [\, x \geq 0, x > y \,], \emptyset\,), (\, x : [\, x < 0, x < z \,], \emptyset\,)\rangle$$

Now, we need to apply each tactic to its corresponding command. We use the function *apply* that takes two lists: the first is a list of functions, and the second is a list of elements to which the functions are applied; it returns the list of the results of applying each function in the first list to the corresponding element in the second list. In our example we have

$$\textit{apply } \langle \textbf{law } \textit{weakPre}(\textit{true}) \,|\, \textbf{law } \textit{assign}(\, x := y + 1\,), \textbf{law } \textit{assign}(\, x := z - 1\,)\rangle$$
$$\langle(\, x : [\, x \geq 0, x > y \,], \emptyset\,), (\, x : [\, x < 0, x < z \,], \emptyset\,)\rangle$$
$$= \langle \textbf{law } \textit{weakPre}(\textit{true}) \,|\, \textbf{law } \textit{assign}(\, x := y + 1\,)\,(\, x : [\, x \geq 0, x > y \,], \emptyset\,),$$
$$\textbf{law } \textit{assign}(\, x := z - 1\,)\,(\, x : [\, x < 0, x < z \,], \emptyset\,)\rangle$$
$$= \langle [\, (\, x : [\, x > y \,], \{x \geq 0 \Rightarrow \textit{true}\}\,), (\, x := y + 1, \{x \geq 0 \Rightarrow y + 1 > y\}\,)\,],$$
$$[\, (\, x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,]\rangle$$

Now, we have to take the distributed cartesian product of this list to get all the

possible combinations of the cells of the first list with the cells of the second list.

$$\Pi \langle\, [\, (\, x : [\, x > y\,], \{x \geq 0 \Rightarrow \mathit{true}\}\,), (\, x := y + 1, \{x \geq 0 \Rightarrow y + 1 > y\}\,)\,],$$
$$[\, (\, x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,]\,\rangle$$

$$= [\, \langle\, (\, x : [\, x > y\,], \{x \geq 0 \Rightarrow \mathit{true}\}\,), (\, x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,\rangle,$$
$$\langle\, (\, x := y + 1, \{x \geq 0 \Rightarrow y + 1 > y\}\,), (\, x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,\rangle\,]$$

The function $\mathit{extractG}$ makes a list of guards of the guarded command.

$$\mathit{extractG} :: \mathit{GuardedComands} \rightarrow \text{pfseq}\ \mathit{Guard}$$
$$\mathit{extractG}\ \ G \rightarrow P = \langle G \rangle$$
$$\mathit{extractG}\ \ G \rightarrow P \ [\!] \ GC = G : (\mathit{extractG}\ \ GC)$$

In the example we have

$$\mathit{extractG}(\, x \geq 0 \rightarrow x : [\, x \geq 0, x = x_0 + 1\,] \ [\!] \ x < 0 \rightarrow x : [\, x < 0, x = x_0 - 1\,]\,) =$$
$$\langle x \geq 0, x < 0 \rangle$$

We combine the list of guards with each element of the list of $RCell$s we get from the distributed cartesian product above. The function $\mathit{insertG}$ takes a list of guards g_i, a list of $RCell$s $(\, p_i, pobs_i\,)$, and returns a pair, where the first element is a guarded command and the second is a set of proof obligations. The guarded command associates each guard g_i to the program p_i: it is $g_1 \rightarrow p_1 \ [\!] \ g_2 \rightarrow p_2 \ [\!] \ ....$ The set of proof obligations is the union of the $pobs_i$.

$$\mathit{insertG} :: \text{pfseq}\ \mathit{Guard} \rightarrow \text{pfiseq}\ \mathit{RCell} \rightarrow$$
$$\text{pfiseq}(\mathit{GuardedCommand}, \mathbb{P}\ \mathit{predicate})$$
$$\mathit{insertG}\ \ \langle\rangle \ (rs) = [\,]$$
$$\mathit{insertG}\ \ gs \ [\,] = [\,]$$
$$\mathit{insertG}\ \ (g : gs)\ ((p, pobs) : rs) = tg\ (g \rightarrow (\mathit{fst}\ r), \mathit{snd}\ r)\ (\mathit{insertG}\ \ gs\ rs)$$

where

$$tg\ (g_1 \rightarrow p_1, pr_1)\ (g_2 \rightarrow p_2, pr_2) = (g_1 \rightarrow p_1 \ [\!] \ g_2 \rightarrow p_2, pr_1 \cup pr_2)$$

We use the function mkGC to apply $\mathit{insertG}\ \langle x \geq 0, x < 0 \rangle$ to each element of

56

the cartesian product above.

$$mkGC :: \text{pfiseq } Guard \rightarrow \text{pfiseq}(\text{pfiseq } RCell) \rightarrow$$
$$\text{pfiseq}(GuardedCommand, \mathbb{P}\ predicate)$$
$$mkGC\ gs = (insertG\ gs)*$$

In our example we have

$$mkGC\ \langle x \geq 0, x < 0 \rangle$$
$$[\, \langle\, (\ x : [\ x > y\,], \{x \geq 0 \Rightarrow true\}\,),$$
$$(\ x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,\rangle,$$
$$\langle\, (\ x := y + 1, \{x \geq 0 \Rightarrow y + 1 > y\}\,),$$
$$(\ x := z - 1, \{x < 0 \Rightarrow z - 1 < z\}\,)\,\rangle\,]$$
$$= [\, (\ x \geq 0 \rightarrow x : [\ x > y\,]\ [\!]\ x < 0 \rightarrow x := z - 1,$$
$$\{x \geq 0 \Rightarrow true, x < 0 \Rightarrow z - 1 < z\}\,),$$
$$(\ x \geq 0 \rightarrow x := y + 1\ [\!]\ x < 0 \rightarrow x := z - 1,$$
$$\{x \geq 0 \Rightarrow y + 1 > y, x < 0 \Rightarrow z - 1 < z\}\,)\,]$$

The last step, which rebuilds the *RCell*s, uses the function Ω_{if}. The arguments of this function are the original set *pobs* of proof obligations and the list of *RCell*s $(\ gc_i, pobs_i\,)$ generated in the previous step. The result is the list of *RCell*s $(\ \mathbf{if}\ \ gc_i\ \mathbf{fi}, pobs\ \cup\ pobs_i\,)$; each guarded command in the argument list is turned into a conditional, and the set of original proof obligations is added to those generated by the tactic.

$$\Omega_{if} :: \mathbb{P}\ predicate \rightarrow \text{pfiseq}(GuardedCommands, \mathbb{P}\ predicate) \rightarrow \text{pfiseq}(RCell)$$
$$\Omega_{if}\ ops = (if\ ops)*$$

where

$$if :: \mathbb{P}\ predicate \rightarrow (GuardedCommands, \mathbb{P}\ predicate) \rightarrow RCell$$
$$if\ ops\ (gc, nps) = (\mathbf{if}\ \ gc\ \mathbf{fi}, ops \cup nps)$$

In our example we have

$$\Omega_{if}\ \{x \geq 1\}$$
$$[\, (\ x \geq 0 \rightarrow x : [\ x > y\,]\ \ [\!]\ x < 0 \rightarrow x := z - 1,$$
$$\{x \geq 0 \Rightarrow true, x < 0 \Rightarrow z - 1 < z\}\,),$$
$$(\ x \geq 0 \rightarrow x := y + 1\ [\!]\ x < 0 \rightarrow x := z - 1,$$
$$\{x \geq 0 \Rightarrow y + 1 > y, x < 0 \Rightarrow z - 1 < z\}\,)\,]$$
$$= [\, (\ \mathbf{if}\ x \geq 0 \rightarrow x : [\ x > y\,]\ \ [\!]\ x < 0 \rightarrow x := z - 1\ \mathbf{fi},$$
$$\{x \geq 0 \Rightarrow true, x < 0 \Rightarrow z - 1 < z, x \geq 1\}\,),$$
$$(\ \mathbf{if}\ x \geq 0 \rightarrow x := y + 1\ [\!]\ x < 0 \rightarrow x := z - 1\ \mathbf{fi},$$
$$\{x \geq 0 \Rightarrow y + 1 > y, x < 0 \Rightarrow z - 1 < z, x \geq 1\}\,)\,]$$

With this example as motivation, we present the definition of the combinator for

the conditional.

$$(\llbracket \boxed{\textbf{if}} \, tacs \, \boxed{\textbf{fi}} \rrbracket \, \Gamma_L \, \Gamma_T)(\textbf{if} \; gc \; \textbf{fi}, pobs) =$$
$$\Omega_{if} \; pobs \; (mkGC \; (extractG \; gc)$$
$$(\Pi(apply \, ((\llbracket \_ \rrbracket \, \Gamma_L \, \Gamma_T) * tacs) \, (\Phi_r * (extractP \; gc))))))$$

Firstly, we extract the commands from the branches gc of the conditional ($extractP$). Secondly, we construct a list of $RCell$s with these commands as their programs, and an empty set of proof obligations ($\Phi_r *$). We apply each element of the list $tacs$ of tactics to the corresponding element in the list of $RCell$s we constructed ($apply \, ((\llbracket \_ \rrbracket \, \Gamma_L \, \Gamma_T) * tacs)$). The next step is to take the distributed cartesian product of the resulting list (Π). Finally, we rebuild the conditionals with the resulting commands ($mkGC \; (extractG \; gc)$) and add the original proof obligations $pobs$ to the new sets of proof obligations ($\Omega_{if} \; pobs$).

The similar definition of the structural combinator $\boxed{\textbf{do}} \, \_ \, \boxed{\textbf{od}}$ is

$$(\llbracket \boxed{\textbf{do}} \, tacs \, \boxed{\textbf{od}} \rrbracket \, \Gamma_L \, \Gamma_T)(\textbf{do} \; gc \; \textbf{od}, pobs) =$$
$$\Omega_{do} \; pobs \; (mkGC \; (extractG \; gc)$$
$$(\Pi(apply \, ((\llbracket \_ \rrbracket \, \Gamma_L \, \Gamma_T) * tacs) \, (\Phi_r * (extractP \; gc))))))$$

The function which generates the list of $RCells$ for the structural combinator $\boxed{\textbf{do}} \, \_ \, \boxed{\textbf{od}}$ is Ω_{do}. Its definition is

$$\Omega_{do} :: \mathbb{P} \; predicate \rightarrow \mathrm{pfiseq}(GuardedCommands, \mathbb{P} \; predicate) \rightarrow$$
$$\mathrm{pfiseq}(RCell)$$
$$\Omega_{do} \; ops = (do \; ops)*$$

where

$$do :: \mathbb{P} \; predicate \rightarrow (GuardedCommands, \mathbb{P} \; predicate) \rightarrow RCell$$
$$do \; ops \; (gc, nps) = (\textbf{do} \; gc \; \textbf{od}, ops \cup nps)$$

The structural combinator $\boxed{\textbf{var}} \, \_ \, \boxed{\rrbracket}$ applies a tactic to the program in a variable block and rebuilds the variable blocks. Its formal definition is

$$(\llbracket \boxed{\textbf{var}} \, t \, \boxed{\rrbracket} \rrbracket \Gamma_L \, \Gamma_T)(\llbracket \, \textbf{var} \; d \bullet p \, \rrbracket, pobs) = (var \; d) * (\llbracket \, t \, \rrbracket \, \Gamma_L \, \Gamma_T \, (p, pobs))$$

The function var rebuilds the variable block after the application of the tactic to its body. It takes as arguments a variable declaration and an $RCell$. The result is a new $RCell$ containing a variable block built from the declaration and the program. The proof obligations are not changed.

$$var : declaration \rightarrow RCell \rightarrow RCell$$
$$var \; d \; (p, pobs) = (\llbracket \, \textbf{var} \; d \bullet p \, \rrbracket, pobs)$$

The structural combinator $\boxed{\textbf{con}} \, \_ \, \boxed{\rrbracket}$ is defined similarly as are all others struc-

tural combinators that handle procedure and variant blocks and parameters. Its definition is

$$([\![\boxed{\mathbf{con}}\ t\,]\!]\!]\Gamma_L\ \Gamma_T)\ ([\![\mathbf{con}\ d \bullet p]\!], \mathit{proofs}) =$$
$$(\mathit{con}\ d) * ([\![t]\!]\Gamma_L\ \Gamma_T\ (p, \mathit{proofs}))$$

where

$$\mathit{con} :: \mathit{declaration} \rightarrow \mathit{RCell} \rightarrow \mathit{RCell}$$
$$\mathit{con}\ d\ (p, \mathit{proofs}) = ([\![\mathbf{con}\ d \bullet p]\!], \mathit{proofs})$$

This function is similar to *var*, as are those below.

The structural combinator $\boxed{\mathbf{pmain}}\_]\!]$ is used to apply a tactic to the main program of a procedure block. Its definition is as follows.

$$(\ [\![\boxed{\mathbf{pmain}}\ t\,]\!]\,]\!]\ \Gamma_L\ \Gamma_T\ )\ (\ [\![\mathbf{proc}\ n \mathrel{\widehat{=}} p_1 \bullet p_2\ ]\!]\,, \mathit{pobs}\ ) =$$
$$(\mathit{procm}\ n\ p_1) * (\ [\![\ t\ ]\!]\ \Gamma_L\ \Gamma_T\ (p_2, \mathit{pobs})\ )$$

where

$$\mathit{procm} : \mathit{name} \rightarrow \mathit{program} \rightarrow \mathit{RCell} \rightarrow \mathit{RCell}$$

$$\mathit{procm}\ n\ p_1\ (p_2, \mathit{pobs}) = (\ [\![\mathbf{proc}\ n \mathrel{\widehat{=}} p_1 \bullet p_2\ ]\!]\,, \mathit{pobs}\ )$$

The structural combinator $\boxed{\mathbf{pmainvariant}}\ \_]\!]$ is used to apply a tactic to the main program of a variant block. Its definition is shown below

$$([\![\boxed{\mathbf{pmainvariant}}\ t\,]\!]\,]\!]\Gamma_L\ \Gamma_T)$$
$$([\![\mathbf{proc}\ \mathit{pname} \mathrel{\widehat{=}} \mathit{body}\ \mathbf{variant}\ v\,\mathbf{is}\ e \bullet p]\!], \mathit{proofs}) =$$
$$(\mathit{variant}\ \mathit{pname}\ \mathit{body}\ v\ e) * ([\![t]\!]\Gamma_L\ \Gamma_T\ (p, \mathit{proofs}))$$

where

$$\mathit{variant} :: \mathit{name} \rightarrow \mathit{program} \rightarrow \mathit{variable} \rightarrow \mathit{expression} \rightarrow \mathit{RCell} \rightarrow \mathit{RCell}$$
$$\mathit{variant}\ \mathit{pname}\ \mathit{body}\ v\ e\ (p, \mathit{proofs}) =$$
$$([\![\mathbf{proc}\ \mathit{pname} \mathrel{\widehat{=}} \mathit{body}\ \ \mathbf{variant}\ \ v\,\mathbf{is}\ e \bullet p]\!], \mathit{proofs})$$

The structural combinator $\boxed{\mathbf{pbody}}\ \_]\!]$ is used to apply a tactic to the procedure body of a procedure block. Its definition is as follows.

$$([\![\boxed{\mathbf{pbody}}\ t\,]\!]\,]\!]\Gamma_L\ \Gamma_T)\ ([\![\mathbf{proc}\ \mathit{pname} \mathrel{\widehat{=}} \mathit{body} \bullet p]\!], \mathit{proofs}) =$$
$$(\mathit{procb}\ \mathit{pname}\ p) * ([\![t]\!]\Gamma_L\ \Gamma_T\ (\mathit{body}, \mathit{proofs}))$$

where

$$\mathit{procb} :: \mathit{name} \rightarrow \mathit{program} \rightarrow \mathit{RCell} \rightarrow \mathit{RCell}$$
$$\mathit{procb}\ \mathit{pname}\ \mathit{main}\ (\mathit{body}, \mathit{proofs}) = ([\![\mathbf{proc}\ \mathit{pname} \mathrel{\widehat{=}} \mathit{body} \bullet \mathit{main}]\!], \mathit{proofs})$$

The structural combinator $\boxed{\mathbf{pbodyvariant}}\ \_]\!]$ is used to apply a tactic to the

procedure body of a variant block. Its definition is shown below.

$$([\![\, \boxed{\textbf{pbodyvariant}}\ t\ ]\!]\!] \Gamma_L\ \Gamma_T)$$
$$([\![\, \textbf{proc}\ \ pname \mathrel{\hat{=}} body\ \textbf{variant}\ \ v \textbf{is}\ e \bullet p]\!], proofs) =$$
$$(variantb\ pname\ v\ e\ p) * ([\![t]\!] \Gamma_L\ \Gamma_T\ (body, proofs))$$

where

$$variantb :: name \rightarrow variable \rightarrow expression \rightarrow program \rightarrow RCell \rightarrow RCell$$
$$variantb\ pname\ v\ e\ main\ (body, proofs) =$$
$$([\![\, \textbf{proc}\ \ pname \mathrel{\hat{=}} body\ \ \textbf{variant}\ \ v\ \textbf{is}\ e \bullet main]\!], proofs)$$

The structural combinator $\boxed{\textbf{pbodymain}}\ t_b\ t_m\,]\!]$ and the structural combinator $\boxed{\textbf{pmainvariantbody}}\ t_b\ t_m\,]\!]$ apply to procedure blocks and variant blocks, respectively. They apply t_b to the body of the procedure, and t_m to the main program. Their definitions are

$$([\![\, \boxed{\textbf{pbodymain}}\ \ t_b\ t_m\ ]\!]\!] \Gamma_L\ \Gamma_T) = \boxed{\textbf{pbody}}\ t_b\,]\!];\ \boxed{\textbf{pmain}}\ t_m\,]\!]$$

and

$$([\![\, \boxed{\textbf{pmainvariantbody}}\ \ t_b\ t_m\ ]\!]\!] \Gamma_L\ \Gamma_T) =$$
$$\boxed{\textbf{pbodyvariant}}\ t_b\,]\!];\ \boxed{\textbf{pmainvariant}}\ t_m\,]\!]$$

When we are not concerned with the type of argument declaration we use the structural combinator $\boxed{\textbf{parcommand}}\ t$. Its definition is

$$([\![\, \boxed{\textbf{parcommand}}\ \ t\,]\!]\,]\!] \Gamma_L\ \Gamma_T)\ ((pars \bullet p), proofs) =$$
$$(parcommand\ pars) * ([\![t]\!] \Gamma_L\ \Gamma_T\ (p, proofs))$$

where

$$parcommand :: declaration \rightarrow RCell \rightarrow RCell$$
$$parcommand\ pars\ (p, proofs) = ((pars \bullet p), proofs)$$

For parameter passing, we have first the semantics of $\boxed{\textbf{val}}\_$. Its definition is

$$[\![\, \boxed{\textbf{val}}\ t\ ]\!]\ \Gamma_L\ \Gamma_T\ ((\textbf{val}\ v : T \bullet p)\,(a), pobs) =$$
$$(val\ v\ T\ a) * ([\![\ t\ ]\!]\ \Gamma_L\ \Gamma_T\ (p, pobs))$$

where

$$val : name \rightarrow Type \rightarrow args \rightarrow RCell \rightarrow RCell$$
$$val\ v\ T\ a\ (p, pobs) = ((\textbf{val}\ v : T \bullet p)(a), pobs)$$

This structural combinator applies the tactic to the body of the parameterized

command and then rebuilds it, using the function *val*.

The structural combinator $\boxed{\mathbf{res}} \_$ is used when we have an argument passed by result in the program of an *RCell*. Its definition is shown below.

$$(\llbracket\boxed{\mathbf{res}}\ t\rrbracket\Gamma_L\ \Gamma_T)\ ((\mathbf{res}\ \ v : T \bullet p)(pars), proofs) =$$
$$(res\ v\ T\ pars) * ((\llbracket t\rrbracket\Gamma_L\ \Gamma_T)\ (p, proofs))$$

where

$$res :: variable \rightarrow type \rightarrow \text{pfseq}\ variable \rightarrow RCell$$
$$res\ v\ T\ pars\ (p, proofs) = ((\mathbf{res}\ v : T \bullet p)(pars), proofs)$$

The structural combinator $\boxed{\mathbf{val\text{-}res}} \_$ is used when we have an argument passed by value-result in the program of an *RCell*. Its definition is as follows.

$$(\llbracket\boxed{\mathbf{val\text{-}res}}\ t\rrbracket\Gamma_L\ \Gamma_T)\ ((\mathbf{val\text{-}res}\ \ v : T \bullet p)(pars), proofs) =$$
$$(valres\ v\ T\ pars) * ((\llbracket t\rrbracket\Gamma_L\ \Gamma_T)\ (p, proofs))$$

$$valres :: variable \rightarrow type \rightarrow \text{pfseq}\ variable \rightarrow\rightarrow RCell$$
$$valres\ v\ T\ pars\ (p, proofs) = ((\mathbf{val\text{-}res}\ v : t \bullet p)(pars), proofs)$$

The tactic $\mathbf{con}\ v \bullet t$ introduces v as a set of variables whose values are taken from an appropriate syntactic class denoted *TERM* below. These variables are used in t as elements of its syntactic class. Their values are angelically chosen so that the tactic succeeds. The semantics is

$$\llbracket\ \mathbf{con}\ v \bullet t\ \rrbracket\ \Gamma_L\ \Gamma_T = \llbracket\ |_{v \in TERM}\ t(v)\ \rrbracket\ \Gamma_L\ \Gamma_T$$

where $|_{v \in TERM}\ t(v)$ is an alternation of all tactics that can be obtained from t, with v ranging over all values in *TERM*. This is an infinite alternation, which can be defined as follows.

$$\Big|_{i=0}^{\infty}\ f(i) = \mu\ X \bullet F(X_0)$$

where $F(X_i) = f(i)\ |\ F(X_{i+1})$. This definition generates the following alternation.

$$f(0)\ |\ f(1)\ |\ \ldots\ |\ f(n)$$

The tactic $\mathbf{applies\,to}\ p\ \mathbf{do}\ t$ needs to consider all ways in which the given program can match p. Its semantics uses the function *equals*, that yields a tactic

that succeeds only if it is presented with a *RCell* matching the given program.

$$equals : RCell \rightarrow Tactic$$

$$equals\ r\ r = [r]$$
$$equals\ r\ h = [\ ],\ \textbf{if}\ h \neq r$$

The definition of the tactic **applies to** p **do** t is

$$[\![\,\textbf{applies to}\ p\ \textbf{do}\ t\,]\!]\,\Gamma_L\,\Gamma_T\,(p_1, pobs) =$$
$$[\![\,\textbf{con}\ p \bullet equals\,(p, pobs);\ t\,]\!]\,\Gamma_L\,\Gamma_T\,(p_1, pobs)$$

For simplicity, we use p, which is a meta-program, as a variable itself (the alternative is to consider the individual variables of p). In $\textbf{con}\ p \bullet equals\,(p, pobs);\ t$, an instantiation of p is angelically chosen to ensure the success of $equals\,(p, pobs);\ t$. The tactic $equals\,(p, pobs)$ succeeds only for those instantiations that match its argument. Above, its argument is $(p_1, pobs)$, so equals $(p, pobs)$ is a filter for the instantiations of p: only those that match p_1 are considered. If there is none, the tactic fails. If a successful instantiation can be chosen, the instantiated values are used in t, which is applied to $(p_1, pobs)$.

2.3.3   Tactic Declarations and Programs

The effect of a tactic declaration is to include a tactic name n in the domain of the tactics environment. This element is mapped to a new function that, for each possible argument $v \in TERM$, gives the semantics of the tactic when the arguments of the tactic are replaced by v. The presence of the clauses **proof obligations** and **program generated** does not change the semantics of a *tacDec*, which is

$$[\![\,\_\,]\!] : tacDec \rightarrow LEnv \rightarrow TEnv \rightarrow TEnv$$

$$[\![\,\textbf{Tactic}\ n(a)\ t\ \textbf{end}\ ]\!]\,\Gamma_L\,\Gamma_T =$$
$$\Gamma_T \oplus \{\,n \mapsto \{\,v \in TERM \bullet v \rightarrow [\![\,t[a \setminus v]\,]\!]\,\Gamma_L\,\Gamma_T\,\}\,\}$$

where $\oplus$ is the overriding operator.

A tactic program is a sequence *tds* of tactic declarations followed by a main tactic t. The semantics is that of t, when evaluated in the environment determined by the tactic declarations and the given laws environment. The definition is as

follows.

$$[\![\, \_ \,]\!] : tacProg \rightarrow LEnv \rightarrow TEnv \rightarrow TEnv$$

$$[\![\, tds\ t\,]\!]\,\Gamma_L\,\Gamma_T = [\![\, t\,]\!]\,\Gamma_L\,(\,decl\ tds\,\Gamma_L\,\emptyset)$$

This environment is defined by the function *decl*.

$$decl : \mathrm{seq}\ tacDec \rightarrow LEnv \rightarrow TEnv \rightarrow TEnv$$

$$decl\,\langle\rangle\,\Gamma_L\,\Gamma_T = \Gamma_T$$
$$decl\,(td_1\ tds)\,\Gamma_L\,\Gamma_T = (\,decl\ tds\,\Gamma_L\,(\,[\![\, td_1\,]\!]\,\Gamma_L\,\Gamma_T\,)\,)$$

In the case that we have an empty tactic declaration, this function returns the tactics environment given as argument. Otherwise, it uses the tactic environment resulting from the first tactic declaration to evaluate the rest of the declarations.

2.3.4 Tacticals

Some tactic languages include a definition of a tactical which makes a robust application of a tactic t. This is called *try* and is defined as

$$try\ t = !(t \mid \textbf{skip})$$

Another tactical makes an exhaustive application of a tactic t. This tactic is defined below.

$$exhaust\ t = (\mu\ Y \bullet (t;\ Y \mid \textbf{skip}))$$

This tactic applies t as many times as possible, terminating (with success) when t fails to apply. An example os its use can be found in Section 2.2.8.

Using the formal semantics presented is this section we are able to prove a set of algebraic laws which can be used for checking tactics equivalence. Furthermore, this set can be used for tactics transformations and optimizations. Next chapter, presents this set of algebraic laws.

Chapter 3

Algebraic Laws

*In this chapter we present laws of reasoning for **ArcAngel**. These laws can be used for checking tactics equivalence. Furthermore, they can be used for tactic transformations and optimizations. In order to prove the completeness of the set of laws presented in this chapter, we use then to support a strategy of reduction to a normal form.*

3.1 Simple Laws

In this section we present laws proposed in the literature for Angel. In Appendix D we prove their soundness in the context of the ArcAngel's semantics. The laws marked with (*) do not hold in the presence of recursive tactics.

3.1.1 Basic properties of composition

The three following laws guarantee that the **skip** is a unit of a sequential composition, and that **fail** is a unit of alternation, and a zero of sequential composition.

> **Law** $1(a)$. **skip**; $t = t$
> **Law** $1(b)$. $t = $ **skip**; t
> **Law** $2(a)$. $t \mid $ **fail** $= t$
> **Law** $2(b)$. $t = $ **fail** $\mid t$
> **Law** $3(a)^*$. t; **fail** $= $ **fail**
> **Law** $3(b)$. **fail** $= $ **fail**; t

Now, we have that sequential composition and alternation are associative, and sequential composition distributes over alternation, on the right only.

> **Law** 4. $t_1 \mid (t_2 \mid t_3) = (t_1 \mid t_2) \mid t_3$
> **Law** 5. t_1; $(t_2;\ t_3) = (t_1;\ t_2)$; t_3
> **Law** 6. $(t_1 \mid t_2)$; $t_3 = (t_1;\ t_3) \mid (t_2;\ t_3)$

The distributive law on the left succeeds only for sequential tactics, which we characterize as follows.

Definition 1.(Sequential Tactics)   A tactic is sequential if it is **skip**, **fail**, **law** l for some law l, or it has the form t_1; t_2 where t_1 is in one of these forms and t_2 is a sequential tactic.

> **Law** 7. t_1; $(t_2 \mid t_3) = (t_1;\ t_2) \mid (t_1;\ t_3)$ for any sequential tactic t_1.

This law establishes that sequential composition distributes over alternation, on the left, if the left tactic is a sequential tactic.

3.1.2 Laws involving Cut

Here, we present some laws involving the *cut* operator, and how this operator interacts with other tactic combinators.

First, we have that atomic law applications and tactics are unchanged by appli-

cations of *cut*.

> **Law** 8. !**skip** = **skip**
> **Law** 9. !**fail** = **fail**
> **Law** 10. !(**law** l ($args$)) =**law** l ($args$)

The use of the cut operator on the left side of a sequential composition allows it to distribute over an alternation on the right.

> **Law** 11. !t_1; ($t_2 \mid t_3$) = (!t_1; t_2) $\mid$ (!t_1; t_3)

The cut operator partially distributes over sequential composition and over alternation.

> **Law** 12. !t_1; !t_2 =!(!t_1; !t_2)
> **Law** 13. !(t_1; t_2) =!(t_1; !t_2)
> **Law** 14(a). !($t_1 \mid t_2$) =!(!$t_1 \mid t_2$)
> **Law** 14(b). !($t_1 \mid t_2$) =!($t_1 \mid$!t_2)

It also has two absorption rules:

> **Law** 15. !($t_1 \mid t_1$; t_2) =!t_1
> **Law** 16. !($t_1 \mid t_2 \mid t_1$) =!($t_1 \mid t_2$)

In the presence of a *cut*, the **skip** becomes a left-zero for alternation, and alternation becomes idempotent.

> **Law** 17. !(**skip** $\mid t$) = **skip**
> **Law** 18. !($t \mid t$) =!t

The *cut* itself is also idempotent.

> **Law** 19. !!t =!t

3.1.3 Laws involving *succs* and *fails*

When we interact **fails** and **succs** with each other and with the other basic tactics, a large set of laws can be proved.

For example, verifying if a tactic t succeeds and then applying it is the same as only applying it. Also, verifying if a tactic t fails and then applying it is the same

as **fail**.

Law 20. **succs** t; $t = t$
Law 21*. **fails** t; $t =$ **fail**

In the presence of **fails**, *cut* becomes identity, as well as **succs**.

Law 22. **fails** $t =$ **fails** $!t =!$ **fails** t
Law 23. **fails**(**succs** t) = **fails** t

The operator **fails** is also associative.

Law 24*. **fails** t_1; **fails** $t_2 =$ **fails** t_2; **fails** t_1

Applying *cut* to an alternation or to a sequential composition can be represented using **fails** and **succs**.

Law 25. $!(t_1 \mid t_2) =!t_1 \mid$ (**fails** t_1; $!t_2$)
Law 26. $!(t_1; t_2) =!(t_1;$ **succs** t_2); $!t_2 \ t_2$)

Applying **succs** and **fails** to an alternation can also be expanded.

Law 27. **succs**$(t_1 \mid t_2) =!($**succs** $t_1 \mid$ **succs** $t_2)$
Law 28. **fails**$(t_1 \mid t_2) =$ **fails** t_1; **fails** t_2

Verifying the success of part of a tactic application is unnecessary if we verify the success of the whole tactic application. However, for failures we should verify each part of the tactic application.

Law 29. **succs** s; **succs**$(s; t) =$ **succs**$(s; t)$
Law 30. **fails** $s =$ **fails** s; **fails**$(s; t)$

We present below another property of the *cut* operator over sequential composition and **fails**.

Law 31. $!s$; **fails** $t =$ **fails**$(!s; t)$; $!s$

Nested applications of **fails** can be removed as follows.

Law 32. **fails**(**fails**$(s; t)$) = **succs** $s \mid$ (**fails** s; **fails** t)
Law 33. **fails**$(s;$ **fails** $t) =$ **fails** $s \mid$ **succs**$(s; t)$

Applying **fails** to the first tactic of a sequential composition and then applying

succs to the whole sequential composition does not succeed.

> **Law** 34*. **fails** s; **succs**$(s;\ t) = $ **fail**

Nested applications of **fails** and **succs** can be removed as follows.

> **Law** 35. **fails**$(t;\ d) = $ **fails**$(t;\ $ **succs** $d)$
> **Law** 36. $!s$; **succs** $t = $ **succs** $(!s;\ t);\ !s$
> **Law** 37. **fails**(**fails** $t) = $ **succs** t

The **fails** and **succs** operators are associative.

> **Law** 38*. **fails** t_1; **succs** $t_2 = $ **succs** t_2; **fails** t_1
> **Law** 39*. **succs** t_1; **succs** $t_2 = $ **succs** t_2; **succs** t_1

It is unnecessary nested applications of **succs**. However, **succs** has no effect when applied to **fails**.

> **Law** 40. **succs**(**succs** $t) = $ **succs** t
> **Law** 41. **succs**(**fails** $t) = $ **fails** t

The following two properties shows, respectively, that the application of **succs** to a sequential composition $t;\ d$ is the same as an application of **succs** to a sequential composition $t;\ $ **succs** d, and that, in the presence of **succs** the *cut* becomes identity.

> **Law** 42. **succs**$(t;\ d) = $ **succs**$(t;\ $ **succs** $d)$
> **Law** 43. **succs** $t = $ **succs** $!t =!$ **succs** t

We present below some properties of composition of **succs** and **fails** with **skip** and **fail**.

> **Law** 44. **succs** **skip** $= $ **skip**
> **Law** 45. **succs** **fail** $= $ **fail**
> **Law** 46. **fails** **skip** $= $ **fail**
> **Law** 47. **fails** **fail** $= $ **skip**

Now, we present some properties of composition and alternation of **fails** and **succs**.

> **Law** 48. **fails** t; **fails** $t = $ **fails** t
> **Law** 49. **succs** t; **succs** $t = $ **succs** t
> **Law** 50*. **succs** t; **fails** $t = $ **fails** t; **succs** $t = $ **fail**
> **Law** 51*. **fails** $t\ \big|\ $ **succs** $t = $ **succs** $t\ \big|\ $ **fails** $t = $ **skip**

Finally, we present some more expanding properties of nested application of **fails**

and **succs**.

> **Law** 52. $\mathbf{succs}(t \mid u) = \mathbf{succs}\ t \mid (\mathbf{fails}\ t;\ \mathbf{succs}\ u)$
> **Law** 53. $\mathbf{succs}(\mathbf{fails}\ s;\ t)) = \mathbf{fails}\ s;\ \mathbf{succs}\ t$
> **Law** 54. $\mathbf{succs}(\mathbf{succs}\ s;\ t) = \mathbf{succs}\ s;\ \mathbf{succs}\ t$
> **Law** 55. $\mathbf{succs}(s;\ \mathbf{fails}\ t) = \mathbf{succs}\ s;\ \mathbf{fails}(s;\ t)$
> **Law** 56. $\mathbf{succs}(s;\ \mathbf{succs}\ t) = \mathbf{succs}\ s;\ \mathbf{succs}\ (s;\ t)$
> **Law** 57. $\mathbf{fails}\ (\mathbf{succs}\ s;\ t) = \mathbf{fails}\ s \mid (\mathbf{succs}\ s;\ \mathbf{fails}\ t)$
> **Law** 58. $\mathbf{fails}(s;\ \mathbf{succs}\ t) = \mathbf{fails}\ s \mid (\mathbf{succs}\ s;\ \mathbf{fails}\ (s;\ t))$

The laws presented in this section are very useful in Section 3.2.

3.1.4 Laws involving Structural Combinators

Here, we have the laws involving the composition of the structural combinator $\boxed{;}$ with sequence, alternation and cut.

> **Law** 59. $(t_1 \boxed{;} t_2);\ (t_3 \boxed{;} t_4) = (t_1;\ t_3) \boxed{;} (t_2;\ t_4)$
> **Law** 60. $t_1 \boxed{;} (t_2 \mid t_3) = (t_1 \boxed{;} t_2) \mid (t_1 \boxed{;} t_3)$
> **Law** 61. $(t_1 \mid t_2) \boxed{;} t_3 = (t_1 \boxed{;} t_3) \mid (t_2 \boxed{;} t_3)$
> **Law** 62. $!(t_1 \boxed{;} t_2) = (!t_1 \boxed{;} !t_2)$

3.1.5 Laws of con

The side-conditions of these laws use the function ϕ, which extracts the set of free-(meta)variables of the tactic to which it is applied. The first property shows that, in a sequential composition $t_1;\ t_2$, if v is not a free-(meta)variable of t_2, we may only use the variables of v in t_1, and vice-versa. However, the last is true only if $t_1 = !t_1$.

> **Law** 63. $(\mathbf{con}\ v \bullet t_1;\ t_2) = (\mathbf{con}\ v \bullet t_1);\ t_2$ provided $v \notin \phi t_2$
> **Law** 64. $(\mathbf{con}\ v \bullet t_1;\ t_2) = t_1;\ (\mathbf{con}\ v \bullet t_2)$ provided $v \notin \phi t_1$ and $!t_1 = t_1$

If v is a not a free-(meta)variable in t, the use of **con** is unnecessary.

> **Law** 65. $(\mathbf{con}\ v \bullet t) = t$ provided $v \notin \phi t$

We can also change the use of a meta-variable v in t for a new meta-variable u if the new meta-variable is not a free-variable in t.

> **Law** 66. $(\mathbf{con}\ v \bullet t) = (\mathbf{con}\ u \bullet t[v \setminus u])$ provided $u \notin \phi t$

In order to prove the completeness of the set of laws presented in this section, we use then to support a strategy of reduction to a normal form in the next section.

3.2   ArcAngel's Normal Form

In this section we prove that tactics written in a subset of ArcAngel that excludes recursion, **abort**, **con**, and **applies to** ... **do** have a unique normal form. This guarantees the completeness of the set of laws presented in the previous section.

First, we present the cut-free normal form for the subset of ArcAngel that includes the operator **law**, sequential composition, alternation, **skip**, **fail**, and the structural combinators. Then, we present the pre-normal form, which expands this subset of ArcAngel and includes the cut(!) operator, **succs**, and **fails**. However, the pre-normal form does not guarantee uniqueness. For this reason, we present the general normal form, which guarantees this property. The idea is to define a normal form for tactics, to show that every tactic can be transformed into a unique normal form using the laws presented in Section 3.1, and proved in Appendix D, and finally to show that if two tactics are equivalent then they have the same normal form.

ArcAngel's normal form provides a complete test whether two tactics texts denotes the same tactic. Furthermore, the laws presented can be used for correctness-preserving transformations, or as justification for automatic tactic optimization [15]. The transformation of tactics to a restricted syntax for special applications can also be obtained [30].

In the following proofs we shall use the notation

$$\Big|_{i:I} t_i$$

to represent the alternate composition $t_{i_1} \mid \ldots \mid t_{i_n}$, for $I = \{i_1, ..., i_n\}$, with the alternation combinators associated to the right. The index sequence I is finite. In the case it has one element only, then the alternation is vacuous and consists only of one instance on the tactic t_i, and in the case I is empty, the expression denotes **fail**.

In the same way, we use the notation

$$;_{i:I} t_i$$

to denote the composition $t_{i_1}; \ldots; t_{i_n}$. The empty sequential composition denotes **skip**.

3.2.1   Cut-free Normal Form

We say that a tactic is in the cut-free normal form if it has the form

$$\Big|_{i:I} \big(;_{j:J_i} \mathbf{M}_j\big)$$

where the $\mathbf{M}_j$ are either **law** l_j, where l_j are names of laws in the domain of the environment of laws, or structural combinators of ArcAngel containing tactics in cut-free normal form.

Lemma 1. Any tactic expressed using basic laws, alternation, sequential composition, **skip**, **fail**, and any structural combinator of ArcAngel can be transformed into a unique tactic in cut-free normal form using the laws presented in Section 3.1.

Proof. The proof is by structural induction over the possible forms of the tactics.

Base cases :
$$\mathbf{skip} = (;\ _\emptyset)$$
$$\mathbf{fail} = |\ _\emptyset$$
$$\mathbf{law}\ l = |\ _{i:I}\ (\mathbf{law}\ l)$$

Inductive step : We assume that
$$t_1 = (\ |\ _{i:I}\ (;\ _{j:J_i}\mathbf{M}_j))\ \text{and}$$
$$t_2 = (\ |\ _{k:K}\ (;\ _{l:L_k}\mathbf{M}_l))$$

We must prove that

a) $t_1\ |\ t_2$ can be put in cut-free normal form

Using the associative Laws 4 and 5 we can easily transform the tactic

$$(\ |\ _{i:I}\ (;\ _{j:J_i}\mathbf{M}_j))\ |\ (\ |\ _{k:K}\ (;\ _{l:L_k}\mathbf{M}_l))$$

to the cut-free normal form.

b) $t_1;\ t_2$ can be put in cut-free normal form

$$(\ |\ _{j:J}\ (;\ _{i:I_j}\mathbf{M}_i));\ (\ |\ _{k:K}\ (;\ _{l:L_k}\mathbf{M}_l))$$

In order to proceed, we use the generalization of the Laws 6 and 7 presented below.

> **Law 6′.** $(\ |\ _{i:I}\ t_i);\ t_3 = (\ |\ _{i:I}\ t_i;\ t_3)$
> **Law 7′.** $t_1;\ (\ |\ _{i:I}\ t_i) = (\ |\ _{i:I}\ t_1;\ t_i)$

And we proceed with the proof

$$(\ |\ _{j:J}\ (;\ _{i:I_j}\mathbf{M}_i));\ (\ |\ _{k:K}\ (;\ _{l:L_k}\mathbf{M}_l))$$
$$=\ |\ _{j:J}\ ((;\ _{i:I_j}\mathbf{M}_i);\ (\ |\ _{k:K}\ (;\ _{l:L_k}\mathbf{M}_l))) \qquad\qquad [\text{Law 6′}]$$
$$=\ |\ _{j:J}\ (\ |\ _{k:K}\ ((;\ _{i:I_j}\mathbf{M}_i);\ ((;\ _{l:L_k}\mathbf{M}_l))) \qquad\qquad [\text{Law 7′}]$$

This last can be put in the cut-free normal form using the associative Laws 4 and 5.

c) The proof for the structural combinators is direct from the definition of the cut-free normal form. For example, $\boxed{\textbf{var}}\, t_1\, \rrbracket$ and $t_1\, \fbox{;}\, t_2$ are already in the cut-free normal form since t_1 and t_2 are in cut-free normal form by assumption.

$\square$

Now, we define when two tactics can be considered equivalent: when they have the same behavior for all *RCells*.

Definition 2.(Tactic Equivalence) Two tactics t_1 and t_2 are equivalent $(t_1 \equiv t_2)$ if for all environment of laws Γ_L, for all environment of tactics Γ_T, and for all *RCell* r, we have that $[\![t_1]\!]\ \Gamma_L\ \Gamma_T\ r = [\![t_2]\!]\ \Gamma_L\ \Gamma_T\ r$.

We prove that if two tactics in normal form are equivalent, then they are syntactically equal.

Lemma 2. If two tactics in normal form are equivalent, then they are identical.
Proof. We prove that there is no point where the tactics differ. We now use the following notation: let the laws which are used in the tactics be l_1, l_2, ..., and assume that the *RCells* are indexed by sequences of law numbers and applications of structural combinators to such sequences, so that:

$$\textbf{law}\ \ l_n\ r_{\langle\rangle} = [r_{\langle n \rangle}]$$
$$\textbf{law}\ \ l_n\ r_l = [r_l{}^\frown{}_{\langle n \rangle}]$$

For alternations we have that:

$$(\textbf{law}\ \ l_n\ \mid\ \textbf{law}\ \ l_m)\ r_l = [r_l{}^\frown{}_{\langle n \rangle}, r_l{}^\frown{}_{\langle m \rangle}]$$

We use the structural combinators to represent themselves. For example, we have that:

$$(\boxed{\textbf{var}}\ \ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3)\, \rrbracket\ \mid\ \textbf{law}\ \ l_4)\ r_{\langle\rangle} = [\,r\, {}_{\boxed{\textbf{var}}\langle 1,2,3\rangle\rrbracket}, r_{\langle 4\rangle}\,]$$

Using this indexing, a tactic applied to $r_{\langle\rangle}$ will produce an account of the tactic's normal form (with the alternation of the sequential composition of the laws in the respective *RCells* in the result list) and so the result is immediate.

$\square$

Conversely, we have that equivalent tactics have a unique normal form.

Theorem 1. Two tactics are equivalent if, and only if, they have the same normal form.

Proof. Let t_{1n} and t_{2n} be the normal form of t_1 and t_2, respectively. First we must prove that $t_1 \equiv t_2 \Rightarrow t_{1n}$ is identical to t_{2n}. Since $t_1 \equiv t_2$ and t_{1n} and t_{2n} are the normal forms of t_1 and t_2 respectively, we can say that $t_1 \equiv t_{1n}$ and $t_2 \equiv t_{2n}$. This is valid because the laws used in the reduction to the normal form are already proved. So, we can also say that $t_{1n} \equiv t_{2n}$, and then, using Lemma 2, that t_{1n} is identical to t_{2n}.

Then we prove that t_{1n} is identical to $t_{2n} \Rightarrow t_1 \equiv t_2$. Since t_{1n} and t_{2n} are the normal forms of t_1 and t_2 respectively, we can say that, $t_{1n} \equiv t_1$ and that $t_{2n} \equiv t_2$. But, we know that, t_{1n} is identical to t_{2n}, and so, $t_1 \equiv t_2$.

$\square$

A set of laws can be said to be complete when tactics which are observationally equivalent (i.e. they behave identically on all goals) are provably so (using the laws of this set).

Corollary 1. The laws in Section 3.1 are complete for the cut-free finite (non-recursive) ArcAngel.

Proof. We have shown that, using the laws given above, we can transform a tactic to its normal form (Lemma 1). Using Theorem 1, we can say that two tactics are equivalent if, and only if, they have the same normal form.

$\square$

3.2.2   Pre-Normal Form

The pre-normal form is for the subset of ArcAngel that includes the subset of the cut-free normal form, the cut (!) operator, and the tactic assertions **succs** and **fails**. First, we revisit the Definition 1 of sequential tactics, so that it also contains the structural combinators.

Definition 3. Sequential Tactics   A tactic is sequential if it is **skip**, **fail**, **law** l for some law l, $!t$, **fails** t or **succs** t, for some sequential tactic t, a structural combinator applied to a sequential tactic, or it has the form $t_1; t_2$, where t_1 is in one of these forms and t_2 is a sequential tactic.

The next Lemma is very useful.

Lemma 3. If a tactic t is a sequential tactic then $t = !t$.

Proof. The proof of this Lemma is presented in Appendix D.6.1.

Now it is possible to prove a completeness result for the finite (non-recursive) Ar-cAngel, including the cut operator. The proof is similar to the one used in the cut-free normal form, but now we have a two-stages process. First, we define a pre-normal form and then we define the general normal form.

The cut-free normal form shows the possible sequences of law applications resulting from a tactic execution. When we include the cut operator in the language, the possibilities depend on the success or failure of sequences of law applications. For this reason we need to include guards which check the validity of the options.

Definition 4. (Pre-normal form) A tactic is in pre-normal form if it has the form

$$\big|_{j:J} g_j; \; (; \; {}_{i:I_j}\mathbf{M}_j)$$

where g_j are guards of the form **succs**($; \; {}_k M_k$) or **fails**($; \; {}_k M_k$), or a possibly empty sequential composition of such guards. The $\mathbf{M}_j$ are either **law** l_j, where l_j are names of laws in the domain of the environment of laws, or structural combinators of ArcAngel containing tactics in pre-normal form.

Now, we want to prove that all the tactics written in the subset of ArcAngel considered can be put in pre-normal form.

Lemma 4. Any tactic written in the language above can be put into pre-normal form using the laws given above.
Proof. In this proof we assume the use of the associative Laws 4 and 5 where necessary. The proof proceeds like the one for the cut-free normal form.

Base cases:
skip = succs skip; $(; \; {}_{i:\emptyset})$

fail $= \big|_{j:\emptyset}$

law $l =$ **succs** (**law** l); **law** l [Law 20]

Inductive step: We assume that
$t_1 = (\big|_{j:J} g_j; \; (; \; {}_{i:I_j}\mathbf{M}_i))$
$t_2 = (\big|_{k:K} g_k; \; (; \; {}_{l:L_k}\mathbf{M}_l))$

We must prove that

a) $t_1 \; \big| \; t_2$ can be put in pre-normal form

74

Using the associative Laws 4 and 5 we can easily transform the tactic

$$= (\mid_{j:J} g_j; \, (; \, _{i:I_j}\mathbf{M}_i)) \mid (\mid_{k:K} g_k; \, (; \, _{l:L_k}\mathbf{M}_l)))$$

to the cut-free normal form.

b) $t_1; \, t_2$ can be put in pre-normal form

$$= (\mid_{j:J} g_j; \, (; \, _{i:I_j}\mathbf{M}_i)); \, (\mid_{k:K} g_k; \, (; \, _{l:L_k}\mathbf{M}_l))$$
$$= \mid_{j:J} ((g_j; \, (; \, _{i:I_j}\mathbf{M}_i)); \, (\mid_{k:K} g_k; \, (; \, _{l:L_k}\mathbf{M}_l))) \qquad \text{[Law 6′]}$$
$$= \mid_{j:J} (!(g_j; \, (; \, _{i:I_j}\mathbf{M}_i)); \, (\mid_{k:K} g_k; \, (; \, _{l:L_k}\mathbf{M}_l))) \qquad \text{[Lemma 3]}$$

We can generalize the Law 11 as
Law 11′. $!t_1; \, (\mid_{i:I} t_i) = \mid_{i:i} t_1; \, t_i$

And so we have

$$\mid_{j:J} (!(g_j; \, (; \, _{i:I_j}\mathbf{M}_i)); \, (\mid_{k:K} g_k; \, (; \, _{l:L_k}\mathbf{M}_l)))$$
$$= \mid_{j:J} (\mid_{k:K} (!(g_j; \, (; \, _{i:I_j}\mathbf{M}_i)); \, (g_k; \, (; \, _{l:L_k}\mathbf{M}_l)))) \qquad \text{[Law 11′]}$$
$$= \mid_{j:J} (\mid_{k:K} (g_j; \, (; \, _{i:I_j}\mathbf{M}_i); \, g_k; \, (; \, _{l:L_k}\mathbf{M}_l))) \qquad \text{[Lemma 3]}$$

This tactic has the form

$$\mid_{i:I} (g_{1_i}; \, t_{1_i}; \, g_{2_i}; \, t_{2_i})$$

Since t_{1_i} are sequences of laws and structural combinators, we have that $t_{1_i} = !t_{1_i}$ (Lemma 3). Now we can apply the laws 31 and/or 36 to assemble the guard components at the beginning of each alternation branch, and remove the *cut* operators by applying again the Lemma 3. For example, consider the tactic

succs (law l_1**); law** l_1**; succs (law** $l_2 \fbox{;}$ **law** l_3**); law** $l_2 \fbox{;}$ **law** l_3
$\quad \mid$ **succs (law** l_4**); law** l_4**; succs (law** l_5**; law** l_6**); law** l_5**; law** l_6
$= $ **succs (law** l_1**); ! law** l_1**; succs (law** $l_2 \fbox{;}$ **law** l_3**); ! law** $l_2 \fbox{;}$ **law** l_3
$\quad \mid$ **succs (law** l_4**); ! law** l_4**; succs (law** l_5**; law** l_6**); !(law** l_5**; law** l_6**)**
$\hfill$ [Lemma 3]
$= $ **succs (law** l_1**); succs(! law** l_1**; law** $l_2 \fbox{;}$ **law** l_3**); ! law** l_1**; !(law** $l_2 \fbox{;}$ **law** l_3**)**
$\quad \mid$ **succs (law** l_4**); succs(! law** l_4**; law** l_5**; law** l_6**); ! law** l_4**; !(law** l_5**; law** l_6**)**
$\hfill$ [Law 36]

$= \textbf{succs} \ (\textbf{law} \ l_1); \ \textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2 \boxed{;} \ \textbf{law} \ l_3); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2 \boxed{;} \ \textbf{law} \ l_3$
  $\big| \ \ \textbf{succs} \ (\textbf{law} \ l_4); \ \textbf{succs}(\textbf{law} \ l_4; \ \textbf{law} \ l_5; \ \textbf{law} \ l_6); \ \textbf{law} \ l_4; \ \textbf{law} \ l_5; \ \textbf{law} \ l_6$

<div align="right">[Lemma 3]</div>

The resulting tactic will be in pre-normal form.

c) $!t$ can be put in pre-normal form.

The tactic $!t$ can be normalized via repeated (if needed) uses of Law 25. As t is in the pre-normal formal (inductive hypothesis) this repeated application of this law will distribute the *cuts* onto the sequential components, from where they can be removed, using Lemma 3, as we can see in

$!(\big|_{j:J} \ g_j; \ (; \ _{i:I_j} \mathbf{M}_i))$
$= !(g_{1_j}; \ (; \ _{i:I_{1_j}} \mathbf{M}_i)) \ \big|$
  $\quad \textbf{fails}(g_{1_j}; \ (; \ _{i:I_{1_j}} \mathbf{M}_i)); \ !(\big|_{j:J \setminus \{1\}} \ g_j; \ (; \ _{i:I_j} \mathbf{M}_i))$ [Law 25]
$= (g_{1_j}; \ (; \ _{i:I_{1_j}} \mathbf{M}_i)) \ \big|$
  $\quad \textbf{fails}(g_{1_j}; \ (; \ _{i:I_{1_j}} \mathbf{M}_i)); \ !(\big|_{j:J \setminus \{1\}} \ g_j; \ (; \ _{i:I_j} \mathbf{M}_i))$ [Lemma 3]

This distribution may result in the creation of nested instances of **succs** and **fails**. Laws 32, 33, and 53-58 can be used to remove those nested instances. At this point, applying the distributive Laws 6 and 11, together with the Lemma 3 will put the tactic in pre-normal form.

For example, consider the tactic

$!(\textbf{succs}(\textbf{law} \ l_1); \ \textbf{law} \ l_1$
  $\big| \ \ \textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2)$
$= !(\textbf{succs}(\textbf{law} \ l_1); \ \textbf{law} \ l_1)$
  $\big| \ \ (\textbf{fails}(\textbf{succs}(\textbf{law} \ l_1); \ \textbf{law} \ l_1); \ !(\textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2))$

<div align="right">[Law 25]</div>

$= (\textbf{succs}(\textbf{law} \ l_1); \ \textbf{law} \ l_1)$
  $\big| \ \ (\textbf{fails} \ (\textbf{law} \ l_1) \ \big|$
      $\quad \textbf{succs}(\textbf{law} \ l_1); \ \textbf{fails}(\textbf{law} \ l_1)); \ (\textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2))$

<div align="right">[Lemma 3]</div>

$= (\textbf{succs}(\textbf{law} \ l_1); \ \textbf{law} \ l_1)$
  $\big| \ \ (\textbf{fails} \ (\textbf{law} \ l_1); \ \textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2)$
  $\big| \ \ (\textbf{succs}(\textbf{law} \ l_1); \ \textbf{fails}(\textbf{law} \ l_1); \ \textbf{succs}(\textbf{law} \ l_1; \ \textbf{law} \ l_2); \ \textbf{law} \ l_1; \ \textbf{law} \ l_2)$

<div align="right">[Law 6]</div>

d) **fails** t can be put in pre-normal form (inductive hypothesis), and so it is an alternation. We can apply a generalized version of Law 28 to the tactic **fails** t to distribute the **fails** through the alternation.

Generalizing Law 28 we have that

Law 28′. $\mathbf{fails}(\mid_{i:I} t_i) =; {}_{i:I} \mathbf{fails}(t_i)$

And so we have

$$\mathbf{fails}(\mid_{j:J} g_j; (; {}_{i:I_j}\mathbf{M}_i))$$
$$=; {}_{j:J} \mathbf{fails}(g_j; (; {}_{i:I_j}\mathbf{M}_i)) \qquad\qquad [\text{Law } 28']$$

Again, the nested instances of **succs** and **fails** can be removed using the Laws 32, 33 and 53-58. In this way, we may introduce alternations as above, and then distributive laws can be used to transform the resulting tactics into pre-normal form.

e) Similar arguments are used to the case **succs** t. The only difference is to change the application of the Law 28 by the application of the Law 52, and to proceed like we did for the **fails** t case.

f) The proofs for the structural combinators are a direct application of the definition of the pre-normal form.

$\square$

In the next subsection we extend the pre-normal form to the general normal form.

3.2.3   General Normal Form

The pre-normal form does not guarantee uniqueness. For example, let s and t be sequence of laws and structural combinators, then the tactics

$$s;\ t$$
$$(\mathbf{succs}\ s);\ s;\ t$$
$$(\mathbf{succs}\ s;\ t);\ s;\ t$$
$$(\mathbf{succs}\ s);\ (\mathbf{succs}\ s;\ t);\ s;\ t$$

are all equivalent, but they are not in the same pre-normal form.

In order to define the general normal form, it is important to introduce some new concepts. Law and structural combinator sequences are sequences of law applications and structural combinators in the form $\mathbf{M}_1; ...; \mathbf{M}_n$, where $\mathbf{M}_i$ are law applications or structural combinators applied to law and structural combinator sequences. For instance, the sequence **law** l_1; (**law** l_2 ; **law** l_4) represents such a sequence. A set T of such sequences is prefix-closed if for any sequence t in T, T

also contains all initial subsequences of t. For example, the set

$$T = \{\mathbf{law}\ l_1, \mathbf{law}\ l_1;\ (\ \mathbf{law}\ l_2\ \boxed{;}\ \mathbf{law}\ l_4\ ), \mathbf{law}\ l_1;\ (\ (\ \mathbf{law}\ l_2;\ \mathbf{law}\ l_3\ )\ \boxed{;}\ \mathbf{law}\ l_4\ )\}$$

is prefix-closed, but the set

$$T = \{\mathbf{law}\ l_1, \mathbf{law}\ l_1;\ (\ (\ \mathbf{law}\ l_2;\ \mathbf{law}\ l_3\ )\ \boxed{;}\ \mathbf{law}\ l_4\ )\}$$

is not, since the element $\mathbf{law}\ l_1;\ (\ \mathbf{law}\ l_2\ \boxed{;}\ \mathbf{law}\ l_4\ )$ is missing.

In the sequel we present the General Normal Form, which is based in the cut-free normal form. The pre-normal form is used in the proof of the lemmas.

Definition 5. (General Normal Form) A tactic t is in general normal form, relative to a law and structural combinator sequence T, if it has the form

$$\Big|_{\ j:J}\ g_j;\ v_j$$

where v_j are tactics in the cut-free normal form and g_j are guards as in the pre-normal form, with certain provisos:

- (Consistency) for each guard g_j, if for some law and structural combinator sequence t, g_j contains **succs** t, it must not contain **fails** s, for any prefix s of t (or t itself);

- (Maximality) for each j, for all t in T, either **succs** t or **fails** t must be present in g_j;

- (Sufficiency) for each j, the success of g_j must be sufficient to guarantee the success of all the alternate clauses in v_j, i.e., if $v_j = \big|_{\ k}\ v_{j_k}$, then for all k, **succs** v_{j_k} must be present in g_j;

- (Mutual Exclusivity) the guards are mutually exclusive; that is, for i and j, with $i \neq j$, there must be some law and structural combinator sequence t for which g_i contains **succs** t and g_j contains **fails** t (or vice-versa).

$\square$

We extend the pre-normal form, imposing the above restrictions on the guards, in order to guarantee uniqueness.

If the conditions above are satisfied we have the following properties.

1. $g_i \neq \mathbf{fail}$: Since the maximality property guarantees that for all t in T, either **succs** t or **fails** t must be present in g_i. So, it can not be **fail**.

2. **succs** t; g_j = g_j or **fails** t; g_j = g_j: Again the maximality property guarantees that all sequence of laws and structural combinators are already guarded in g_i, either with **succs** or **fails**. The consistency property guarantees that if there is **succs** t, there must not be **fails** s, for s some prefix of t. In both cases, using the Laws 24, 39, 38, 48, and 49 we can eliminate the first guard **succs** t or **fails** t and maintain only the guards g_j.

3. **succs** g_j = **succs** g_j; **succs** v_{j_k}: Since the sufficiency property afirms that **succs** v_{j_k} is a consequence of **succs** g_j.

4. g_i; g_j = **fail** $(i \neq j)$: Since the mutual exclusivity property says that for two different guards g_i and g_j $(i \neq j)$ there is some law and structural combinator sequence t for which g_j contains **succs** t and g_j contains **fails** t (or vice-versa). So, if **succs** t skips in g_i, then **fails** t fails (or vice-versa), then we use the Laws 1, 2 and 3.

For example, consider two mutual excludents guards

$$\textbf{succs}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2)$$
$$\textbf{fails}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2)$$

if we compose these two guards, we have

$\textbf{succs}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2); \textbf{ fails}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2)$
$= \textbf{succs}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2); \textbf{ fails}(\textbf{law } l_2)$ [Law 24]
$= \textbf{succs}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_1); \textbf{ fails}(\textbf{law } l_2)$ [Law 48]
$= \textbf{fail}; \textbf{ fails}(\textbf{law } l_2)$ [**succs** t = **skip** $\Rightarrow$ **fails** t = **fail**]
                                      [**succs** t = **fail** $\Rightarrow$ **fails** t = **skip**]
$= \textbf{fail}$ [Law 3]

Using the conditions seen above, we can say that a tactic in the general normal form has, for example, the form

$$...$$
$$\mid \quad ...; \textbf{ succs } t; \; ...; \; v_n$$
$$...$$
$$\mid \quad ...; \textbf{ fails } t; \; ...; \; v_m$$
$$...$$

Using the Laws 24, 39, and 38 we can re-order the components of the guards of

these tactics to the form

$$\dots$$
$$\mid \quad \textbf{succs} \ t; \ \dots; \ v_n$$
$$\dots$$
$$\mid \quad \textbf{fails} \ t; \ \dots; \ v_m$$
$$\dots$$

This property, together with the mutual exclusion property, means that the outer-most alternation can also be re-ordered.

$$\dots$$
$$\mid \quad \textbf{succs} \ t; \ \dots; \ v_n$$
$$\mid \quad \textbf{fails} \ t; \ \dots; \ v_m$$
$$\dots$$

Therefore, an arbitrary pair of adjacent alternation branches of a tactic in general normal form can be considered to have the form

$\textbf{succs} \ t; \ s_1 \ \mid \ \textbf{fails} \ t; \ s_2$
$= (\textbf{fails} \ t \ \mid \ \textbf{succs} \ t); \ (\textbf{succs} \ t; \ s_1 \ \mid \ \textbf{fails} \ t; \ s_2) \quad$ [Laws 51 and 1]
$= (\textbf{fails} \ t; \ \textbf{succs} \ t; \ s_1)$
$\quad \mid \ (\textbf{fails} \ t; \ \textbf{fails} \ t; \ s_2)$
$\quad \mid \ (\textbf{succs} \ t; \ \textbf{succs} \ t; \ s_1)$
$\quad \mid \ (\textbf{succs} \ t; \ \textbf{fails} \ t; \ s_2) \quad\quad\quad\quad\quad$ [Laws 6 and 11,Lemma 3]
$= \textbf{fails} \ t; \ s_2 \ \mid \ \textbf{succs} \ t; \ s_1 \quad\quad\quad\quad\quad\quad$ [Laws 50,2,48 and 49]

So, we can say that the ordering of the tactic may be changed arbitrarily. This means that the normal forms achieved below is unique only modulo these two forms of reordering. This relative definition is important since the guards g_i of two tactics, for example, t_1 and t_2, can be syntactically different, but equivalent, given the commutative laws on **succs** and **fails**, and so, the tactics are equivalent.

Lemma 5. Given a sufficiently large prefix-closed set of law and structural combinator sequences, T, any tactic in pre-normal form can be put into a unique general normal form, relative to T (unique modulo reordering of the guards), using laws drawn from the set given above. T is sufficiently large if contains at least the minimal set of law and structural combinator sequences determined by consideration of the tactic t in pre-normal form: T must contain all the instances of law and structural combinator sequences.

Proof. The idea behind this proof is to sequentially compose an alternation of

guards which is equivalent to **skip** with the tactic in pre-normal form (since it is equivalent to the tactic itself by Law 1), and then to transform it into a tactic in general normal form.

First, consider the tactic formed from all possible guards for law and structural combinator sequences in T:

$$; \, _{t:T}(\textbf{succs} \; t \; \mid \; \textbf{fails} \; t)$$

This is equivalent to **skip** since one of the branches will succeed and behave like **skip**. Using the distributive Laws 6 and 11 (applying Lemma 3 when necessary), we can transform this expression into

$$\mid \, _{i:I} \; g_i$$

where $g_i = \textbf{succs} \; t$ or $g_i = \textbf{fails} \; t$ for all $t : T$.

It is still equal to **skip** because the laws preserve the functionality of the tactics. Using Law 34 and the commutative Laws 24, 39, and 38, we can remove all those i from I for which g_i is equivalent to **fail**. Let us name this new I as I'.

To illustrate these first steps let us have $T = \{l_1, l_1; \; l_2, l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4\}$. We have the tactic formed from all possible guards for law and structural combinator sequences in T as seen below:

$$(\textbf{succs} \; l_1 \; \mid \; \textbf{fails} \; l_1);$$
$$(\textbf{succs} \; l_1; \; l_2 \; \mid \; \textbf{fails} \; l_1; \; l_2);$$
$$(\textbf{succs} \; l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4 \; \mid \; \textbf{fails} \; l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$$

Using the sequential composition and alternation distributive laws we get the following alternation.

$\textbf{succs} \; l_1; \; \textbf{succs} \; (l_1; \; l_2); \; \textbf{succs}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[1]
$\mid \; \textbf{succs} \; l_1; \; \textbf{fails} \; (l_1; \; l_2); \; \textbf{succs}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[2]
$\mid \; \textbf{fails} \; l_1; \; \textbf{succs} \; (l_1; \; l_2); \; \textbf{succs}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[3]
$\mid \; \textbf{fails} \; l_1; \; \textbf{fails} \; (l_1; \; l_2); \; \textbf{succs}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[4]
$\mid \; \textbf{succs} \; l_1; \; \textbf{succs} \; (l_1; \; l_2); \; \textbf{fails}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[5]
$\mid \; \textbf{succs} \; l_1; \; \textbf{fails} \; (l_1; \; l_2); \; \textbf{fails}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[6]
$\mid \; \textbf{fails} \; l_1; \; \textbf{succs} \; (l_1; \; l_2); \; \textbf{fails}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[7]
$\mid \; \textbf{fails} \; l_1; \; \textbf{fails} \; (l_1; \; l_2); \; \textbf{fails}(l_1; \; l_2; \; l_3 \,\boxed{;}\, l_4)$	[8]

The branches [2],[3],[4] and [7] can be removed using Laws 34, 24, 39, and 38.

Finally, we get

$$\mid \ \textbf{succs} \ l_1; \ \textbf{succs} \ (l_1; \ l_2); \ \textbf{succs}(l_1; \ l_2; \ l_3 \,\fbox{;}\, l_4) \qquad\qquad [1]$$

$$\mid \ \textbf{succs} \ l_1; \ \textbf{succs} \ (l_1; \ l_2); \ \textbf{fails}(l_1; \ l_2; \ l_3 \,\fbox{;}\, l_4) \qquad\qquad [5]$$

$$\mid \ \textbf{succs} \ l_1; \ \textbf{fails} \ (l_1; \ l_2); \ \textbf{fails}(l_1; \ l_2; \ l_3 \,\fbox{;}\, l_4) \qquad\qquad [6]$$

$$\mid \ \textbf{fails} \ l_1; \ \textbf{fails} \ (l_1; \ l_2); \ \textbf{fails}(l_1; \ l_2; \ l_3 \,\fbox{;}\, l_4) \qquad\qquad [8]$$

This final tactic has the mutual exclusion property mentioned in the definition of the general normal form. It also has the consistency property, since those branches which were equal to **fail** were removed, and the maximality property, in that for each t in T, for each i, either g_i contains **fails** t or **succs** t.

Proceeding with the proof, if we sequentially compose the tactic we worked out above with the tactic in pre-normal form we have

$$(\mid_{\ i:I'} \ g_i); \ (\mid_{\ j:J_i} \ h_j; \ (;\ _{k:K_j} M_k))$$

Applying the distributive laws we have

$$\mid_{\ i:I'} \ \mid_{\ j:J_i} \ g_i; \ h_j; \ (;\ _{k:K_j} M_k)$$

This tactic is still equivalent to the original tactic in pre-normal form.

By the maximality property, and the sufficient size of T, for each h_j, g_i pair, we have either that each component of h_j is present in g_i, and so, by the commutative laws and Laws 48 and 49, $g_i;\ h_j = g_i$, or that h_j has a **succs** t for which g_i has a **fails** t, and so, by the commutative laws and Law 34, we have that $g_i;\ h_j = \textbf{fail}$. Using the commutative laws and Law 2, we can rewrite the tactic, removing the branches from J_i (J_i' replaces J_i) which are equal to **fail**.

$$\mid_{\ i:I'} \ \mid_{\ j:J_i'} \ g_i; \ (;\ _{k:K_j} M_k)$$

The next step is to omit (using the commutative laws and Law 2) the inner alternation branches whose guards contains an instance of **fails**$(;\ _k M_k)$, since this branch will fail (J_i'' replaces J_i'). In the case J_i'' becomes empty, and so denotes **fail**, this i can be omitted from I' using Law 2.

Finally, the distributive law can be used to transform the tactic into the required normal form:

$$\mid_{\ i:I'} \ (g_i; \ \mid_{\ j:J_i''} \ (;\ _{k:K_j} M_k))$$

This is in normal form, since the g_i remain maximal, consistent, and mutually exclusive. Their sufficiency arises as a result of the maximality and the omission (in the last step, above) of clauses which must fail.

$$\square$$

Lemma 6. Two tactics in general normal form (relative to some sufficiently large prefix-closed set of rule and structural combinator sequences T) are equivalent if, and only if, they are identical modulo reordering of the guards.

Proof. The first part of this proof is to show that if two tactics are identical modulo reordering of the guards, then they are equivalent. This is trivial since the Laws 24, 38, and 39 makes reordering possible and the laws are sound.

The second part of the proof is to show that, if two tactics in general normal form are equivalent, then they are identical modulo reordering of the guards. We assume that the laws and structural combinators we refer to are $M_1, M_2, ..., M_n$ and the *RCells* are decorated with pairs. The first element of this pair is a prefix-closed set of law and structural combinator sequences. These sequences are those which can succeed when applied to that *RCell*. The second element is the law and structural combinator sequence which contains the laws and the structural combinators which have been already successfully applied.

This is

$$M_i \; r_{s,t} \neq [\,] \Leftrightarrow \exists \, xs : s \;\; \big| \;\; head \; xs = M_i.$$

and

$$M_i \; r_{s,t} = r_{s',t} \smallfrown_{\langle r_i \rangle} \; where \; s' = xs \;\; \big| \;\; r_i : xs \in s$$

otherwise the laws application fails.

Analyzing the behavior of a tactic applied to different *RCells*, we can determine the guard/body (in cut-free normal form) pairs of its normal form. The application of a tactic to an *RCell* $r_{s,\langle\rangle}$ will either result in the empty sequence or in a sequence of the form $r_{s_1,t_1}, ..., r_{s_n,t_n}$. The former case does not interest us since it will not help us to reconstruct the tactic in general normal form.

In the latter case, the body part of an alternation can be reconstructed as the alternation of $t_1, ..., t_n$. The guard for that branch can be also reconstructed from **succs** applied to each member of s and **fails** applied to each member of $T \setminus s$.

If we take all the initials *RCells* $r_{s,\langle\rangle}$ (with s any prefix-closed subset of T) for which the outcome is not the empty sequence, it is possible to reconstruct the whole tactic in normal form.

For example, let us consider an hypothetical case where the set

$$T = \{l_1, l_1; \; l_2, l_1; \; l_3, l_1; \; l_2; \; l_3 \boxed{;} \, l_4\}$$

and there are only two possible initial *RCell* which are

$$R_{\langle l_1, l_1; \; l_2, l_1; \; l_3, l_1; \; l_2; \; l_3 \boxed{;} \, l_4 \rangle, \langle\rangle}$$
$$R_{\langle l_1, l_1; \; l_2, l_1; \; l_2; \; l_3 \boxed{;} \, l_4 \rangle, \langle\rangle}$$

Now, we consider that when we apply a tactic t_1 to these two *RCells* we get,

83

respectively,

$$\langle R_{\langle l_3\rangle, \langle l_1, l_2, l_3 \boxed{;} l_4\rangle}, R_{\langle l_2, l_2;\ l_3 \boxed{;} l_4\rangle, \langle l_1, l_3\rangle}\rangle$$
$$\langle R_{\langle\rangle, \langle l_1, l_2, l_3 \boxed{;} l_4\rangle}\rangle$$

If we rebuild this tactic as described before, we have, for the first branch (from the first *RCell*)

$$\mathbf{succs}(l_1);\ \mathbf{succs}(l_1;\ l_2);\ \mathbf{succs}(l_1;\ l_3);$$
$$\mathbf{succs}(l_1;\ l_2;\ l_3 \boxed{;} l_4);\ (l_1;\ l_2;\ l_3 \boxed{;} l_4\ \mid\ l_1;\ l_3)$$

fails does not appear because $T \setminus s$ is empty for the first *RCell*. For the second *RCell* we have

$$\mathbf{succs}(l_1);\ \mathbf{succs}(l_1;\ l_2);\ \mathbf{succs}(l_1;\ l_2;\ l_3 \boxed{;} l_4);\ \mathbf{fails}(l_1;\ l_3);\ (l_1;\ l_2;\ l_3 \boxed{;} l_4)$$

Now, we make an alternation of these to branches, and we have the tactic in the normal form.

$$\mathbf{succs}(l_1);\ \mathbf{succs}(l_1;\ l_2);\ \mathbf{succs}(l_1;\ l_3);$$
$$\mathbf{succs}(l_1;\ l_2;\ l_3 \boxed{;} l_4);\ (l_1;\ l_2;\ l_3 \boxed{;} l_4\ \mid\ l_1;\ l_3)$$
$$\mid$$
$$\mathbf{succs}(l_1);\ \mathbf{succs}(l_1;\ l_2);\ \mathbf{succs}(l_1;\ l_2;\ l_3 \boxed{;} l_4);\ \mathbf{fails}(l_1;\ l_3);\ (l_1;\ l_2;\ l_3 \boxed{;} l_4)$$

We guarantee the maximality property by ensuring that every member of T is present in each guard, since we apply **succs** to each member of s and **fails** to each member of $T \setminus s$. The mutual exclusivity is ensured by the fact that the sets s we consider in each initial *RCell* $r_{s,\langle\rangle}$ are different, and so, the set of sequences with **succs** are different. The consistency property is ensured by the fact that the guards either contains **succs** x if $x \in s$ or **fails** x if $x \in T \setminus s$. The last property, the sufficiency, is guaranteed by the fact that the law and structural combinators sequences $t_1, ..., t_n$ are those which succeeds when applied to the initial *RCell*. So, these law and structural combinator sequences are in s, and consequently, **succs** t_i is in the guard g_j.

In this way, a tactic in normal form can be completely characterized (modulo reordering) by the set of goals on which it succeeds and the outcomes when it does so, and the result follows immediately.

$$\square$$

Theorem 2. Two tactics are equivalent under all law and structural combinators instantiations if, and only if, they have the same general normal form, modulo reordering of the guards.

Proof. Let t_{1n} and t_{2n} be the normal form of t_1 and t_2, respectively. First we must prove that $t_1 \equiv t_2 \Rightarrow t_{1n}$ is identical to t_{2n}, modulo reordering of the guards. Since $t_1 \equiv t_2$ and t_{1n} and t_{2n} are the normal forms of t_1 and t_2 respectively, we can say that $t_1 \equiv t_{1n}$ and $t_2 \equiv t_{2n}$. This is valid because all the laws used in the reduction to the normal form are already proved. So, we can also say that $t_{1n} \equiv t_{2n}$, and then, using Lemma 6, that t_{1n} is identical to t_{2n}, modulo reordering of the guards.

Then we prove that t_{1n} is identical to t_{2n}, modulo reordering of the guards, implies $t_1 \equiv t_2$. Since t_{1n} and t_{2n} are the normal forms of t_1 and t_2 respectively, we can say that, $t_{1n} \equiv t_1$ and that $t_{2n} \equiv t_2$. But, we know that, t_{1n} is identical to t_{2n}, modulo reordering of the guards, and so, $t_1 \equiv t_2$.

$\square$

Corollary 2. The set of laws is complete for the language described above.

Proof. The proof of this corollary is similar to the proof of the Corollary 1.

$\square$

In order to put tactics which involve **tactic** t in the general normal form, we can replace the tactic call by the tactic body itself. It must be in the general normal form, or it must be possible to reduce it to the general normal form.

We are yet to consider tactics involving recursion, **abort**, **con**, and **applies to** ... **do**. This is left as further work.

Chapter 4

Gabriel: a tool for **ArcAngel**

In this chapter we present Gabriel. We present its concepts, discuss its user interface, and present the constructs of ArcAngel available in Gabriel. Finally, we present an example of writing and using a tactic.

4.1   **Gabriel** Concepts

Gabriel [27] is a tactic editor plugged-in to Refine [8], a tool that supports program development using Morgan's refinement calculus. Using Refine, the user starts with a formal specification, and repeatedly applies laws of refinement in order to obtain the program which correctly implements the initial formal specification. After each law application, Refine shows to the user the program development, the collected code and the proof obligations generated with the law applications. Refine also allows users to save a program development in order to continue it later. Furthermore, Refine also has the undo and redo operations.

Gabriel is activated from Refine. Using Gabriel, the user can:

- Create a tactic: the user writes a tactic in ASCII ArcAngel(see Section 4.3);

- Edit a tactic;

- Generate a tactic: the system verifies the syntax correctness, and includes the tactic to the list of tactics of Refine. Afterwards, the user can apply the generated tactic in program developments;

- Remove a tactic;

- Apply a tactic.

The generation of a tactic automatically from a program development in Refine is also a required functionality. However, this is not implemented in the current version of Gabriel and is left as future work.

Some diagrams of Gabriel implementation can be found in Appendix E. We present the class diagrams of the Gabriel-Refine integration and tactics hierarchy, and the sequence diagrams for tactic generation and application.

4.2   **Gabriel**'s User Interface

Gabriel has a simple user interface. This interface is presented in Figure 4.1. The buttons descriptions are (from left to right):

- Start new tactic;

- Open existing tactic;

- Save tactic;

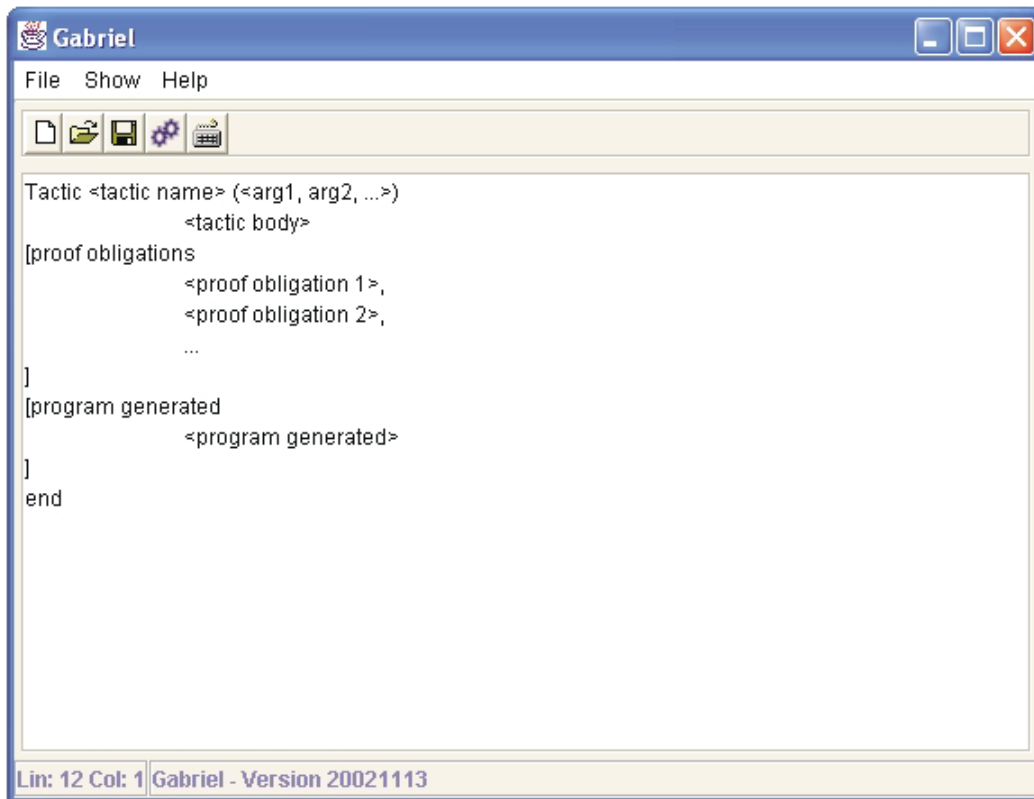- Generate tactic and insert it into the tactics list of Refine;

Figure 4.1: Gabriel's User Interface

- Open symbols keyboard, which is used to insert the ASCII representation of ArcAngel's constructs.

Refine's user interface was modified in order to use Gabriel. Basically, the changes are: the creation of the tactics list and the addition of a button that opens Gabriel. Figure 4.2 shows the tactics list window.

Gabriel's user interface is based [24] on the LUCID, the User-Centered Interface Design [16]. First, we made an action-object analysis; this activity consists of building a directed-graph describing the activities of program refinements, and building an object-tree containing the objects of Refine-Gabriel's user interface. Using this technique we analyzed the correspondence of program refinement activities and the user-interface of Refine and Gabriel.

Finally, a test was made with Refine-Gabriel's potential users. This test consisted of a tactic creation and application using Gabriel and Refine, respectively. After the tests, we interviewed the users, identifying usage difficulties and the reasons for them. Based on the whole test, the users identified positive points of Gabriel and made suggestions for improvement. The positive aspects pointed were:

- Easy to open the tactic editor (Gabriel);

- Easy to see the result of a tactic creation;

- Easy application of laws and tactics;

- Nice integration Refine-Gabriel

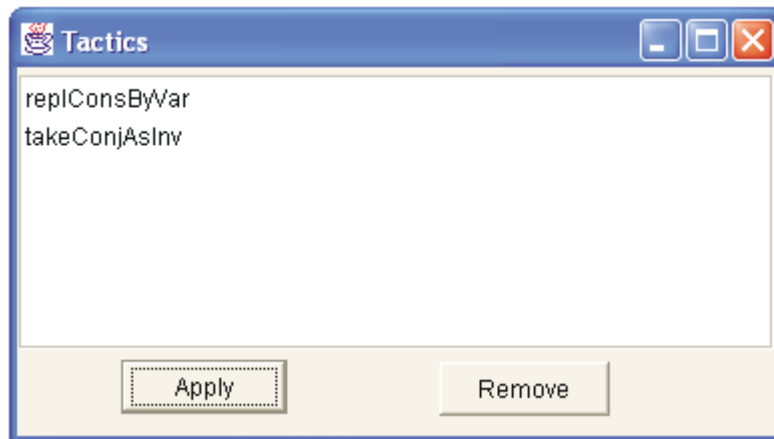Figure 4.2: Refine's Tactics List

The suggestions made were the inclusion of:

- A symbol keyboard for Gabriel;

- ArcAngel documentation in the help;

- A Generate tactic button in Gabriel;

- A Show tactic/law details facility with a double click on the tactic/law in the tactic/law list in Refine;

- A tactic template to start new tactics.

Most of these suggestions were applied to Refine and Gabriel.

4.3   Constructs of Gabriel

Gabriel uses ArcAngel as a tactic language. However, some constructs of ArcAngel, as structural combinators, cannot be written in ASCII. For this reason, Gabriel uses an ASCII representation of ArcAngel which is presented in Appendix F.1. The
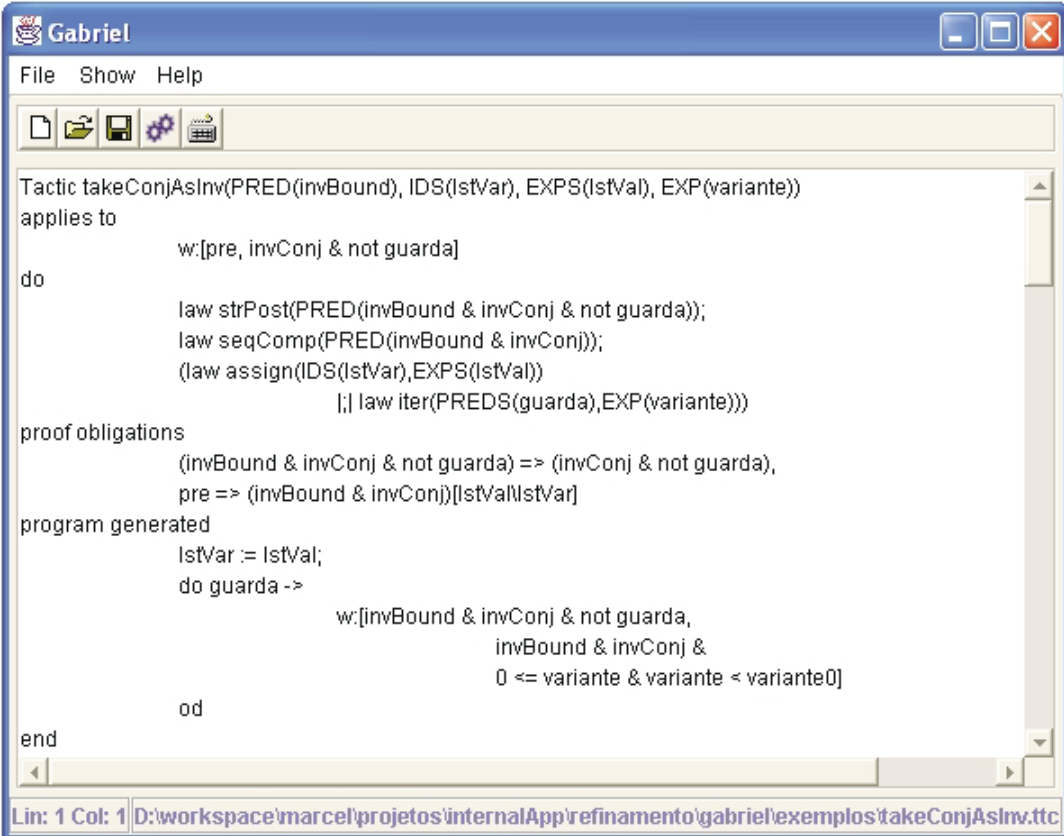
tactics **con**, μ, $\boxed{\textbf{val}}$, $\boxed{\textbf{res}}$, and $\boxed{\textbf{val-res}}$ are not implemented in the current version of Gabriel. These are left as future work.

Furthermore, Gabriel supports most of the laws of Morgan's refinement calculus. The names of the laws implemented in Refine and their usage template are presented in Appendix F.2. The law usage template defines the name of the law and its arguments. For example, the strengthening post-condition law must be used as $strPost(PRED(newPost))$, where $strPost$ is the name of the law and $PRED(newPost)$ is the argument of the law.

All the arguments are typed. The types that can be used in Gabriel are presented in Appendix F.3. The list of tactics as an argument is not implemented in this version of Gabriel and is left as future work.

4.4   Using **Gabriel**

In this section we present an example of a tactic creation and application. We use


```
Tactic takeConjAsInv(PRED(invBound), IDS(lstVar), EXPS(lstVal), EXP(variante))
applies to
            w:[pre, invConj & not guarda]
do
            law strPost(PRED(invBound & invConj & not guarda));
            law seqComp(PRED(invBound & invConj));
            (law assign(IDS(lstVar),EXPS(lstVal))
                        |;| law iter(PREDS(guarda),EXP(variante)))
proof obligations
            (invBound & invConj & not guarda) => (invConj & not guarda),
            pre => (invBound & invConj)[lstVal\lstVar]
program generated
            lstVar := lstVal;
            do guarda ->
                        w:[invBound & invConj & not guarda,
                                    invBound & invConj &
                                    0 <= variante & variante < variante0]
            od
end
```

Lin: 1 Col: 1 D:\workspace\marcel\projetos\internalApp\refinamento\gabriel\exemplos\takeConjAsInv.ttc

Figure 4.3: *takeConjAsInv* written in ASCII ArcAngel

the tactic *takeConjAsInv*(Section 2.2.3) and the example presented in page 23.

In order to create a tactic we must open Gabriel pressing Gabriel's button in Refine. Afterwards, we must write the tactic *takeConjAsInv* in ASCII ArcAngel as seen in Figure 4.3.

After writing the tactic, we must save it before its generation. We can save a tactic pressing the save button or choosing the save option in the file menu of Gabriel.

After saving the tactic we can generate it pressing the generate button or choosing the generate option in the file menu of Gabriel. If the generation succeeds, the tactic is inserted in the list of tactics of Refine. However, if some syntax error is found, the system shows an error message indicating the line of the error.

Once the tactic is generated we can apply it in Refine. In our example, the program $q, r : [a >= 0 \, \& \, b > 0, (a = q*b + r \, \& \, 0 <= r) \, \& \, not \, r >= b]$ is the initial program. We start the program by pressing the start new development button. Figure 4.4 presents Refine's window used to insert the new start specification.
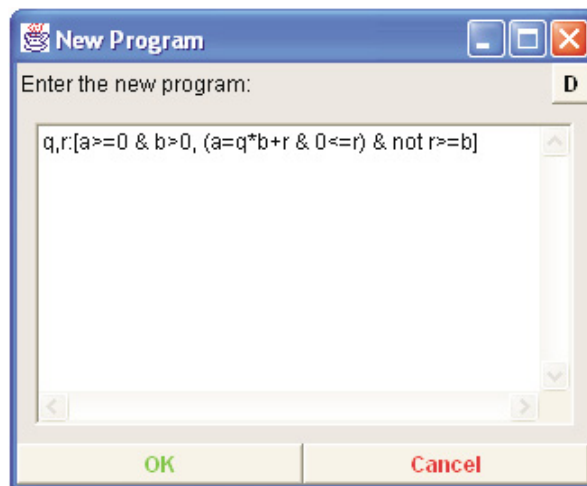

Figure 4.4: Refine's New Development Window

Then, we select the program in the development window and the tactic *takeConjAsInv* in the tactics list window and press the apply button in this window. The system

will require the parameters of the tactic application, as presented in Figure 4.5.
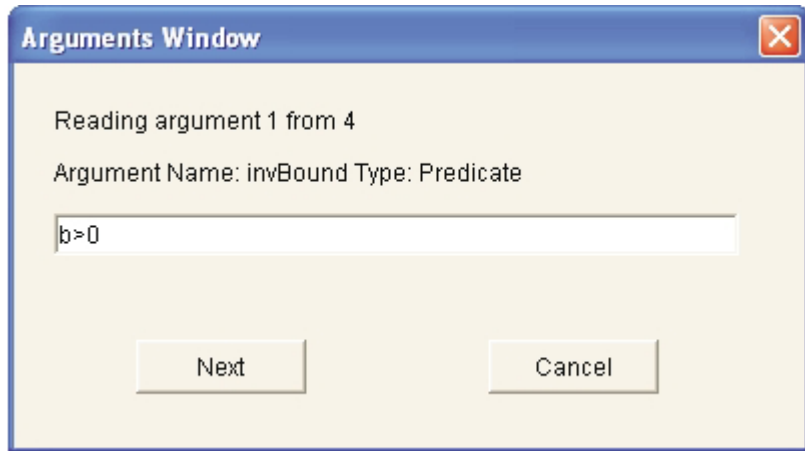

Figure 4.5: **Gabriel**'s Arguments Window

We insert the values presented in Table 4.1.
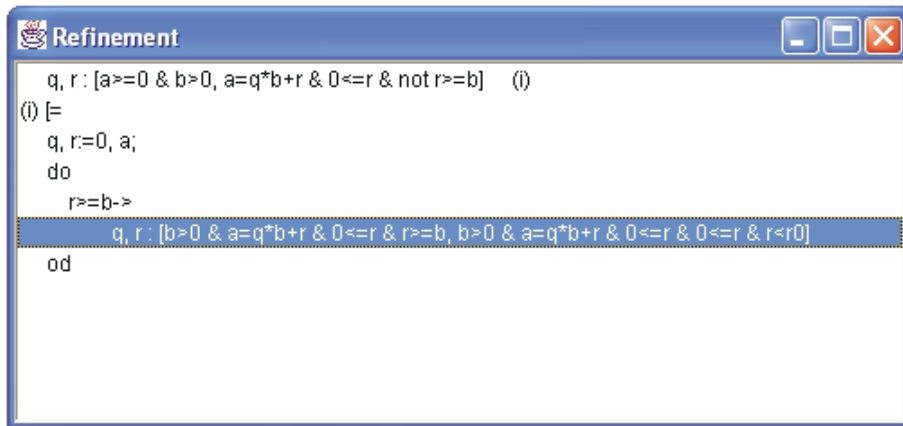
Figure 4.6: Refine's Development Window

After the insertion of the last argument value, Refine actually applies the tactic. The program development window, the proof obligations window and the collected code window are actualized. Figure 4.6 presents the resulting development window.
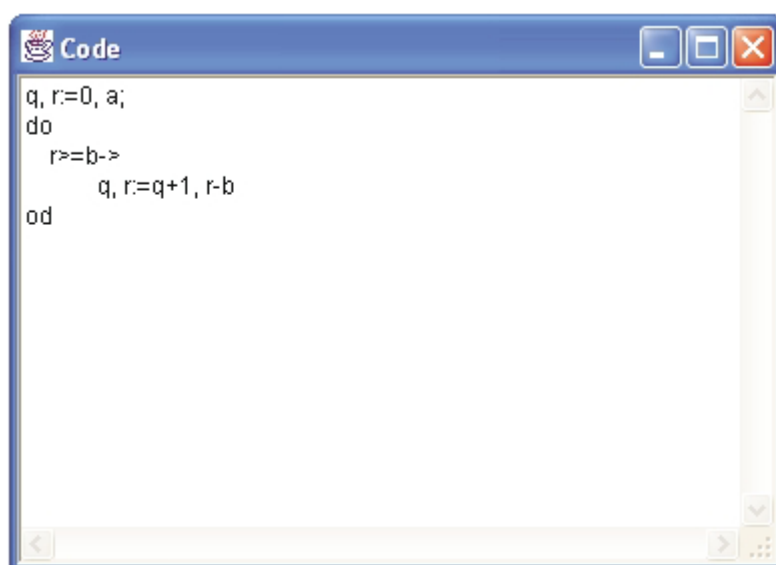
Finally, we select the guarded program in the iteration by clicking the left-button of the mouse on it, select the assignment with initial variables law in the list of laws window, and apply it by pressing the apply button of this window. We use the

arguments q, r and $q + 1, r - b$.

Argument Name	Argument's Value
invBound	$b > 0$
lstVar	q, r
lstVal	$0, a$
variante	r

Table 4.1: Argument's Values

At the end of the development, the resulting collected code window is that

Figure 4.7: Refine's Collected Code Window

presented in Figure 4.7.

Chapter 5

Conclusions

In this chapter we give an overview of the results of our work. Related works are also presented in this chapter. Finally, we suggest topics for future work.

5.1 Contributions and Conclusions

We have presented ArcAngel, a refinement-tactic language. Using this language, it is possible to specify commonly used strategies of program development. Tactics can be used as transformation rules; this shortens developments and improves their readability. We have identified some strategies of program development, documented them as tactics using ArcAngel, and used them in some program developments as single transformation rules.

We have defined the semantics of ArcAngel based on the Angel semantics. The main difference is that Angel is a very general language and its goals have no defined structure. For us, a goal is called an *RCell*, which is a pair with a program as its first element and a set of proof obligations as its second element. The application of a tactic to an *RCell* returns a list of *RCell*s containing the possible output programs, with their corresponding set of proof obligations.

We have shown the soundness of algebraic laws for reasoning about ArcAngel tactics. We have covered most of the laws that have been proposed for Angel. However, most of them have no available proof in the literature. These proofs are provided here in the context of ArcAngel. Since these laws are valid, the strategy proposed to reduce finite Angel tactics to a normal form has been applied to ArcAngel tactics.

Finally, we have presented Gabriel , a tool to support the use of ArcAngel. We have presented the concepts of the tool and briefly discussed its user interface. We have presented the constructs of ArcAngel available in Gabriel, the names of the laws of Morgan's refinement calculus available in ArcAngel, and the types of arguments which can be used in the tool. An example of a tactic creation and application has also been presented.

5.2 Related Work

Several other tools that support program refinement are available in the literature. Some of them provide facilities for use of tactics, but, as far as we know, none of them has a special purpose language, like ArcAngel.

The aim of the work and the tool described in [11] is also to find a way of encoding developing strategies and reusing them in other developments. As Gabriel their system helps the user to develop programs using the refinement calculus. It supports the specification and programming constructs described in [23]. The user loads an initial program (or types it), and then, repeatedly uses commands displayed in a menu. Basically, the user can apply a refinement law or a tactic (refine command), undo the last refinement step, elide a part of the program, or expand to see hidden parts again. The expand and print command produces a LaTeX description of the refinement step. The rerun command is also supported

by this system. The interaction between the user and the system is menu-mouse based.

This system supports data refinement of variable blocks. The user specifies the abstract variables, the concrete variables, and the coupling invariant that defines the relationship between them. It uses an assisted, theory-driven rewriter to discharge proof obligations. This is possible only if they can be simplified to true, otherwise the user is asked to prove it. Also, trivial predicates are automatically removed from specification statements. This facility is still not supported by Refine, nevertheless their system does not display rules and proof obligations as Refine does.

Tactics in their system can be written in a language that is embedded in Prolog, which has a formal semantics [18, 1]. As far as we know, however, the extension of Prolog used as a tactic language does not have a well defined semantics. This language has operators for sequential, alternate, and conditional composition. The sequential and the alternate composition are as in ArcAngel, and the conditional composition corresponds to ArcAngel's cut operator. Constructs for tackling parts of the programs are also provided. Nevertheless, their tactic language does not have a recursion operator.

Their tactic language has a different style if compared with ArcAngel's. At each step of the tactic, the user defines the part of the program that is being refined, the refinement law, and the structure of the resulting program. In ArcAngel, the user must only define the refinement law and to which point of the program it applies. The structure of the resulting program is contained in the law definition. Although the former gives an easier-to-read tactic, the latter allows tactic programs that can be implemented using parallel application of tactics. For instance, to implement **if** t_1 t_2 **fi** , we can apply t_1 and t_2 in parallel.

Primitive refinement rules can also be defined using the same language. This brings to the user the possibility to include new refinement laws in the system. Nevertheless, it is improbable that new laws beyond those in Morgan's refinement calculus are needed in any development.

The Refinement Editor [33, 34] has an embedded tactic language. A fundamental feature of the editor is a record of law applications in a program development, and its reuse in later developments. The Refinement Editor is a multiple windows system containing a main window for the overall development; a window for the refinement of a sub-program; a window for the proof obligations; a window for the list of available commands; and a window that displays the record of law applications. The main window accepts commands and displays the transformations. The user can save and load a development, cut and paste parts of a development, modify existing law applications, and label segments of the development. The proof obligations can be discharged using an theorem prover that needs to be provided by the user, but may be invoked directly from the editor. The support for data refinement is still to be done.

The record of law applications uses a tactic language similar to ArcAngel. It has constructs for sequential composition and structural combinators. Nevertheless, the Refinement Editor tactic language does not have operators for alternation, recursion and backtracking. The tactic language also does not have a formal semantics. The library of basic laws contains the laws of Morgan's refinement calculus. This library can be extended with user-defined laws. Their definitions use pre-defined laws and must provide the provisos and the parameters of the laws. This brings advantages and disadvantages as already discussed.

The Proxac system [31, 32] is a very general transformation editor. Its goal is to support the application of a sequence of transformation steps based on algebraic rules of one or more theories. As such, Proxac supports both refinement by calculation and theorem proving. Proxac offers a reduce option, which applies a predefined set of transformations exhaustively. Some of the main functions are: apply a rule (possibly with a hint to the system), delete the last step, and manipulate expressions.

Its user-interface is based on windows, and uses mouse and keyboard for the interaction with the user. By selecting the line to which the rule is going to be applied, selecting a rule in the rules window, and pressing the a-key, the rule is applied to the selected line. The editor attempts to match it against the left-hand side or the right-hand side of the rule. If they do not match, then the editor proceeds by attempting to match a sub-expression of the selected line. The system attempts to verify the side conditions. If it fails, then the editor questions their validity. The user must decide whether to proceed or not.

Proxac theories are organized in modules. A module has a name, and declares a list of items. These items are declarations of identifiers, operators, imports of other modules, and statements of a property or a rule. Those are written using a specific language described in [31]. A transformation session occurs in the context of the definitions in one or more modules. Proxac contains a module with all the laws of Morgan's refinement calculus, as well as modules with pre-defined operators and properties.

Proxac does not have a tactic language, and also does not provide the reuse of previous derivations. The user can only save the derivation to edit it later.

The aim of the tool presented in [12, 13] is to support more flexible styles of program developments than the strict top-down. In particular, program construction by incremental enhancement and program reuse is considered.

The tool has two windows: the program window, where the user can select components of the program to be transformed by clicking and dragging with the mouse; and the script window, which is divided in three parts. The center part displays the refinement step that is currently being assembled or ready to be applied. The top part contains the history of steps that have been applied (script). The bottom part contains the steps that are available to be applied.

At each step of the derivation, the user can perform several operations on the script window: insert a new refinement step; step backwards through previous steps; jump to any earlier step; apply the next step from the script, possibly modifying a step; delete the next step of a script; insert an existing script into the script window; and apply remaining steps from the script automatically, until a step fails. The modifications can be propagated to later steps in a script. Adaptations use an editor which has only a query-replace facility. The refinement tool automatically constructs a script describing the derivation. These derivation scripts can be constructed, saved, copied, and edited.

The script language used to record the derivations is an easy to learn tactic language also based in Prolog terms. Its only construct is the sequential composition. It has the same style as [11]. We have not found in the literature a definition of the script language formal semantics.

The tool also attempts to discharge proof obligations using a library of rewrite rules. Nevertheless, the tool allows the derivations to proceed if it cannot discharge the proof obligations. Data refinement rules are available. Procedures can be introduced extracting a fragment of the script, inserting a corresponding procedure declaration, and replacing the removed fragment by a procedure call.

In [4, 35, 36] the HOL System, a theorem-proving assistant that can be used to formalize theories and verify proofs within these theories, is used in program refinements. The user interacts with the HOL System through ML, a functional programming language, making definitions and evaluating expressions. Definitions and expressions are written in a combination of predicate calculus and typed lambda calculus. All this requires quite an amount of ML programming.

Using this language, the user can program tactics that can be used in later developments. The combinators available are THEN, a sequential composition; REPEAT, which repeats a tactic as long as it is applicable; and ORELSE, which tries a second tactic if the first one fails.

The formalization of the refinement concepts includes a theory of commands. A number of algorithmic and data refinement rules have also been proved. The specification language formalized is more restricted than the refinement calculus language of Back and Wright [3]. For example, multiple assignments and recursion cannot be expressed in this specification language. Beyond the sequential, assignment, alternation, iteration commands, and variable and logical constants blocks, Back and Wright refinement calculus language presents also non-deterministic assignment (angelical and demonic), and inverse commands. Procedures and parameters are not supported by their language, although their language can be extended to support them.

In order to prove a refinement, the programmer must express the problem in their formalization. The proof mainly consists of rewriting using basic definitions.

The Program Refinement Tool (PRT) [6] is built as an extension of the Ergo

theorem prover [5]. It supports a language similar to Morgan's refinement calculus.

PRT uses the window inference proof paradigm, which is based on term rewriting [29]. In this paradigm, the proof is conducted with a stack of windows. Each window has a focus, which is the expression to be transformed; a set of assumptions from the context of the window; a relation between the focus and the expression to which it is tranformed; and a theorem which relates the focus of the window to that of the window that originated it (the parent window). In order to transform a subexpression, the user opens a subwindow using special inference rules, called window rules. This pushs a new window onto the stack of windows. When the transformation of a window is finished, the user closes the window using a window rule. This pops the window from the stack and uses its theorem to infer how to transform the parent window.

The program window inference theory of PRT has support for dealing with program variables, applying refinement rules, and managing the kind of context that arises during refinement. This support includes a definition of the refinement relation, the refinement rules, window opening and closing rules, a mechanism for handling program variables and their substitution, a tactic support for applying and instantiating refinement rules, and a proof interface. Using PRT, each refinement step is an application of an instance of a theorem schema in the theory inserted in the tool, with the meta variables instantiated to correspond to the current context.

The interface of PRT is a window divided in several frames. The most important frames are the proof frame, which contains the panes that display the current state of the refinement; the proof script frame, where the commands required to perform the refinement are recorded; the rules frame, which controls the display and selection of matching rules; and the help frame. To select a subterm of the proof or a law to be applied, the user can point and click with the mouse. This opens a frame on the subterm, the proof browser, or applies the selected rule to the highlighted term.

Some proof obligations are discharged using the Ergo window inference theory, which includes logic, set theory, arithmetic, and structures such as sequences. The remaining proof obligations are handled in one of three ways. The default is to record them as conjectures, which must be discharged before closing the current window. Another possibility is to record them as postulates in the theory, which can be discharged later as separated proofs. Finally, they can be discharged as they are generated. The theorem and its proof are stored and can be viewed subsequently using the browser and applied in subsequent developments.

Tactics support is in the form of proof scripts written in a language based on Prolog. It includes commands for all the rule applications and the proofs of their side-conditions as sub-proofs. The user can copy, combine, and edit scripts. As mentioned above, PRT records proof scripts as developments. We can also load a script and replay it line by line to recreate the proof. We have not found a detailed description of this language.

New refinement rules can be postulated and proved with the definition of refinement and the weakest precondition semantics of the program constructs used. Application theories can be defined building upon an existing library that includes first-order predicate logic, arithmetic, and set theory.

Jim Grundy [14] uses the HOL System to formalize a refinement support system. Instead of formalizing the refinement calculus based on weakest pre-conditions, he formalizes programs as predicates. This has a goal-oriented reasoning approach, which is a popular method of proving theorems. However, this is not usual when we are refining programs, since we do not know the final implementation of the specification before finishing its refinement.

In his formalization, refinements preserve only partial correctness with respect to a specification. This means that a program implements correctly a specification if it stops. Otherwise, it may run indefinitely.

Specifications are written using logic, with a extension to support initial variables. The executable constructs are skip, multiple assignment, alternation, sequential composition, and recursion. These constructs and the support for initial variables are a syntactic sugaring of standard notations supplied by an user-interface. The laws formalized in the system are skip introduction, assignment, alternation, iteration, and sequential composition.

The user creates tactics as sets of commands that automate, or partially automate, common program refinement steps. Users can add these tactics in order to use them in later developments.

5.3 Future Work

Generalizing the normal form to encompass the rest of the language is not trivial and is still to be done. Recursion, **abort**, **con**, and **applies to** ... **do** are still to be inserted in the subset of ArcAngel that has a defined unique normal form. Moreover, the simple approach for expression arguments must also be extended. These should be evaluated before being used.

The automatic tactic generation from a program development in Refine is not implemented in the current version of Gabriel. Besides, Gabriel does not implement a subset of ArcAngel. This subset includes recursion (μ), **con** $v \bullet t$, and the structural combinators $\boxed{\textbf{val}}$, $\boxed{\textbf{res}}$, and $\boxed{\textbf{val-res}}$. The inclusion of the structural combinators in Gabriel is very simple. However, the inclusion of recursion and **con** $v \bullet t$, and the automatic tactic generation needs more complex analysis and implementation.

Furthermore, in Gabriel tactics cannot be passed as argument. The implementation of tactics as arguments allows the implementation of some tactics which are not yet implementable in Gabriel [25]. However, this needs a theoretical analysis which is still to be done. Moreover, the functions *tail*, *head'*, and *seqToList*, used

to define the tactic *procCalls*(Section 2.2.8) and the tactic *procArgs*(Section 2.2.9), are not implemented in Gabriel. Their implementation makes it possible their use in Gabriel and, consequently, the use of the tactic *recProcArgs*(Section 2.2.10). The implementation of these functions in Gabriel is very simple.

Gabriel's error messages in tactic applications do not express precisely the reasons of the error. This happens to allow backtracking in these applications. In the current implementation of Gabriel, if we display a tactic application failure, further applications are not possible anymore. For this reason, when a tactic application in Gabriel fails, an error message is given to the user indicating only that an error has occurred. However, the reasons of the failure are not displayed. In order to give the user some feedback about the application errors, an application log is generated and indicated by the error message. Implementing a way to display the application log is very simple and is left as future work.

The discharge of trivial proof obligations is not implemented in Refine. Using an existing theorem-prover to discharge them is also required. Finally, generating a LaTeX document describing a program development and a tactic can also be inserted in Refine and Gabriel, respectively.

Appendix A

ArcAngel's Novel Constructs

Here, we distinguish the tactics of ArcAngel which are inherited from Angel and the tactics of ArcAngel which are not defined in Angel. In the figure below, we reproduce part of the syntax of ArcAngel and indicate on the right the constructs that we already available in Angel.

$$
\begin{aligned}
tactic \quad ::= \quad & \textbf{law}\ \ name\ args \\
\mid\ & \textbf{tactic}\ \ name\ args \\
\mid\ & \textbf{skip}\ \mid\ \textbf{fail}\ \mid\ \textbf{abort} && [\textbf{Angel's Constructs}] \\
\mid\ & tactic;\ tactic\ \mid\ tactic \mid tactic && [\textbf{Angel's Constructs}] \\
\mid\ & !\,tactic\ \mid\ \mu\,name \bullet tactic && [\textbf{Angel's Constructs}] \\
\mid\ & \textbf{succs}\ tactic\ \mid\ \textbf{fails}\ tactic && [\textbf{Angel's Constructs}] \\
\mid\ & tactic\ \boxed{;}\ tactic \\
\mid\ & \boxed{\textbf{if}}\ tactic^{+}\ \boxed{\textbf{fi}}\ \mid\ \boxed{\textbf{do}}\ tactic^{+}\ \boxed{\textbf{od}} \\
\mid\ & \boxed{\textbf{var}}\ tactic\ \boxed{]\!]}\ \mid\ \boxed{\textbf{con}}\ tactic\ \boxed{]\!]} \\
\mid\ & \boxed{\textbf{pmain}}\ tactic\ \boxed{]\!]}\ \mid\ \boxed{\textbf{pmainvariant}}\ tactic\ \boxed{]\!]} \\
\mid\ & \boxed{\textbf{pbody}}\ tactic\ \boxed{]\!]}\ \mid\ \boxed{\textbf{pbodyvariant}}\ tactic\ \boxed{]\!]} \\
\mid\ & \boxed{\textbf{pbodymain}}\ tactic\ tactic\ \boxed{]\!]} \\
\mid\ & \boxed{\textbf{pmainvariantbody}}\ tactic\ tactic\ \boxed{]\!]} \\
\mid\ & \boxed{\textbf{val}}\ tactic\ \mid\ \boxed{\textbf{res}}\ tactic\ \mid\ \boxed{\textbf{val-res}}\ tactic \\
\mid\ & \boxed{\textbf{parcommand}}\ tactic \\
\mid\ & \textbf{con}\ v \bullet tactic && [\textbf{Angel's Construct}] \\
\mid\ & \textbf{applies to}\ \ program\ \textbf{do}\ tactic
\end{aligned}
$$

Figure A.1: ArcAngel's Novel Constructs

Appendix B

Infinite Lists

We present the model for infinite lists adopted here [19]. The set of the finite and partial sequences of members of X is defined as

$$\text{pfseq}\, X ::= \mathsf{partial}\langle\!\langle \text{seq}\, X \rangle\!\rangle \mid \mathsf{finite}\langle\!\langle \text{seq}\, X \rangle\!\rangle$$

We define an order $\sqsubseteq$ on these pairs such that for $a, b : \text{pfseq}\, X$, if a is finite, then $a \sqsubseteq b$ if, and only if, b is also finite and equal to a. If a is partial, then $a \sqsubseteq b$ if, and only if, a is a prefix of b.

$$\_\sqsubseteq\_ : \text{pfseq}\, X \leftrightarrow \text{pfseq}\, X$$

$\forall\, gs, hs : \text{seq}\, X\, \bullet$
  $\mathsf{finite}\, gs \sqsubseteq \mathsf{finite}\, hs \Leftrightarrow gs = hs$
  $\mathsf{finite}\, gs \sqsubseteq \mathsf{partial}\, hs \Leftrightarrow false$
  $\mathsf{partial}\, gs \sqsubseteq \mathsf{finite}\, hs \Leftrightarrow gs\ \mathsf{prefix}\ hs$
  $\mathsf{partial}\, gs \sqsubseteq \mathsf{partial}\, hs \Leftrightarrow gs\ \mathsf{prefix}\ hs$

A chain of sequences is a set whose elements are pairwise related.

$$chain : \mathbb{P}(\mathbb{P}(\text{pfseq}\, X))$$

$$\forall\, c : \mathbb{P}(\text{pfseq}\, X) \bullet c \in chain \Leftrightarrow (\forall\, x, y : c \bullet x \sqsubseteq y \vee y \sqsubseteq x)$$

The set pchain contains all downward closed chains.

$$pchain : \mathbb{P}\, chain[X]$$

$$\forall\, c : chain[X] \bullet c \in pchain \Leftrightarrow (\forall\, x : c;\ y : \text{pfseq}\, X \mid y \sqsubseteq x \bullet y \in c)$$

The set pfiseq contains partial, finite, and infinite list of elements of X, which are

prefixed-closed chains of elements in pfseq X.

$$\text{pfiseq}\, X == pchain[X]$$

The idea is that $\bot = \mathsf{partial}\,\langle\rangle$, the empty list $[\,] = \mathsf{finite}\,\langle\rangle$, and the finite list $[e_1, e_2, \ldots, e_n]$ is represented by the set containing $\mathsf{finite}\,\langle e_1, e_2, \ldots, e_n\rangle$ and all approximations to it. An infinite list is represented by an infinite set of partial approximations to it. The infinite list itself is the least upper bound of such a set.

The definitions of the functions used in this thesis are as follows.

1. The map function $*$ maps a function f to each element of a possibly infinite list.

$$* : (X \rightarrow Y) \rightarrow \text{pfiseq}\, X \rightarrow \text{pfiseq}\, Y$$

$\forall\, c : \text{pfiseq}\, X;\ f : X \rightarrow Y\ \bullet$
     $f * \bot = \bot$
     $f * c = \{\, x : c \bullet pfmap\, f\, x\,\}$

The function $pfmap$ maps the function f to the second element of x.

$$pfmap : (X \rightarrow Y) \rightarrow \text{pfseq}\, X \rightarrow \text{pfseq}\, Y$$

$\forall\, xs : \text{seq}\, X;\ f : X \rightarrow Y\ \bullet$
     $pfmap\, f\,(\mathsf{finite}\, xs) = \mathsf{finite}\,(f \circ xs) \wedge$
     $pfmap\, f\,(\mathsf{partial}\, xs) = \mathsf{partial}\,(f \circ xs)$

2. The distributed concatenation returns the concatenation of all the elements of a possibly infinite list of possibly infinite lists.

$$\overset{\infty}{\curlywedge}\!/ : \text{pfiseq}(\text{pfiseq}\, X) \rightarrow \text{pfiseq}\, X$$

$\forall\, s : \text{pfiseq}(\text{pfiseq}\, X)\ \bullet$
     $\overset{\infty}{\curlywedge}\!/\, s = \bigsqcup_{\infty} \{\, c : s \bullet \overset{\infty}{\curlywedge}\!/\, c\,\}$

It uses the function $\overset{\infty}{\curlywedge}\!/$, which is the distributed concatenation for $\text{pfiseq}(\text{pfiseq}\, X)$.

The function *cat* is the standard concatenation function for X^*.

$\overset{\infty}{\frown}/ : \mathrm{pfseq}(\mathrm{pfseq}\, X) \to \mathrm{pfseq}\, X$

$\overset{\infty}{\frown}/(f, \langle\rangle) = \emptyset$
$\overset{\infty}{\frown}/(p, \langle\rangle) = (p, \langle\rangle)$
$\overset{\infty}{\frown}/(f, \langle g\rangle) = [g]$
$\overset{\infty}{\frown}/(p, \langle g\rangle) = [g] ^\wedge \{(p, \langle\rangle)\}$
$\overset{\infty}{\frown}/(f, gs \frown hs) = (\overset{\infty}{\frown}/(f, gs)) \bowtie (\overset{\infty}{\frown}/(f, hs))$
$\overset{\infty}{\frown}/(f, gs \frown hs) = (\overset{\infty}{\frown}/(f, gs)) \bowtie (\overset{\infty}{\frown}/(p, hs))$

The function $\bowtie$ is the concatenation function for possibly infinite lists. Its definition is

$\_ \bowtie \_ : \mathrm{pfseq}\, X \times \mathrm{pfseq}\, X \to \mathrm{pfseq}\, X$

$\forall\, a, b : \mathrm{pfseq}\, X \bullet$
    $a \bowtie b = \{\, x : a;\ y : b \bullet x ^\wedge y \,\}$

where the function $^\wedge$ is the concatenation function for pfseq X defined as

$\_ ^\wedge \_ :: \mathrm{pfseq}\, X \times \mathrm{pfseq}\, X \to \mathrm{pfseq}\, X$

$\forall\, gs, hs : \mathrm{seq}\, X;\ s : \mathrm{pfseq}\, X \bullet$
    $\mathsf{finite}\ gs ^\wedge \mathsf{finite}\ hs = \mathsf{finite}\,(gs \frown hs)$
    $\mathsf{finite}\ gs ^\wedge \mathsf{partial}\ hs = \mathsf{partial}(gs \frown hs)$
    $\mathsf{partial}\ gs ^\wedge s = \mathsf{partial}\ gs$

3. The function *head′* returns a list containing the first element of a possibly infinite list.

   $head' : \mathrm{pfseq}\, X \to \mathrm{pfseq}\, X$

   $head'\ xs = take\ 1\ xs$

   It uses the function *take* that returns a list containing the first n elements of a possibly infinite list.

   $take : \mathbb{N} \to \mathrm{pfseq}\, X \to \mathrm{pfseq}\, X$

   $take\ n\ \bot = \bot$
   $take\ 0\ xs = [\,]$
   $take\ n\ [\,] = [\,]$
   $take\ n\ xs = [head\ xs] \frown (take\ (n-1)\ (tail\ xs))$

   For a list $(x : xs)$, the function *head* returns x and the function *tail* returns xs. For a pair (a, b), the function *first* returns a and the function *second* returns b.

4. The function $^\circ$ applies a possibly infinite list of functions to a single argument.

$$\_^\circ : \mathrm{pfiseq}(X \nrightarrow Y) \rightarrow X \rightarrow \mathrm{pfiseq}\ Y$$

$$\bot^\circ\ x = [\,]$$
$$[\,]^\circ\ x = [\,]$$
$$[f]^\circ\ x = [f\ x]$$
$$(\mathit{fs} \frown \mathit{gs})^\circ\ x = \mathit{fs}^\circ\ x \frown \mathit{gs}^\circ\ x$$

5. The function Π is the distributed cartesian product for possibly infinite lists.

$$\Pi : \mathrm{seq}(\mathrm{pfiseq}\ X) \rightarrow \mathrm{pfiseq}(\mathrm{seq}\ X)$$

$$\Pi\langle xs\rangle = e2l * xs$$
$$\Pi(xs : xss) = [(a : as) \mid a \leftarrow xs,\ as \leftarrow \Pi\ xss]$$
where
$$e2l\ x = [x]$$

Appendix C

Refinement Laws

We present the refinement laws of Morgan's refinement calculus. For each law, we present its name, the arguments used for its application, the transformation of the program to which it is applied, and the provisos of its application.

Law $strPost(post_2)$.
$w : [pre, post_1] \sqsubseteq w : [pre, post_2]$
provided $post_2 \Rightarrow post_1$

Law $weakPre(pre_2)$.
$w : [pre_1, post] \sqsubseteq w : [pre_2, post]$
provided $pre_1 \Rightarrow pre_2$

Law $assign(w := E)$.
$w : [pre, post] \sqsubseteq w := E$
provided $pre \Rightarrow post[w \setminus E]$

Law $simpleEspec()$.
$w : [w = E] = w := E$
provided E does not contain w

Law $absAssump()$. $\{pre'\}w : [pre, post] = w : [pre' \wedge pre, post]$

Law $skipIntro()$. $w, x : [pre, post] \sqsubseteq$ **skip** provided $w = w_0 \wedge pre \Rightarrow post$

Law $seqComp(mid)$.
$w : [pre, post] \sqsubseteq w : [pre, mid]; \ w : [mid, post]$
provided mid and $post$ have no free initial variables.

Law $skipComp()$.
For any program p
skip; $p = p$; **skip** $= p$

Law $fassign(x := E)$.
$w, x : [pre, post] \sqsubseteq w, x : [pre, post[x \setminus E]]; \ x := E$

107

Law $alt(\langle G_0, ..., G_n \rangle)$. $w : [pre, post] \sqsubseteq$ **if** $(?\ i.G_i \rightarrow w : [G_i \wedge pre, post])$ **fi**
    provided $pre \Rightarrow G_0 \vee ... \vee G_n$

Law $altGuards(\langle H_0, ..., H_n \rangle)$.
    Let $GG = G_0 \vee \ldots \vee G_n$ and $HH = H_0 \vee \ldots \vee H_n$.
    if $(?\ i.G_i \rightarrow w : [G_i \wedge pre, post])$ **fi** $\sqsubseteq$
        if $(?\ i.H_i \rightarrow w : [H_i \wedge pre, post])$ **fi**
    provided $GG \Rightarrow HH$ and for each i, $GG \Rightarrow (H_i \Rightarrow G_i)$

Law $strPostIV(post_2)$.
    $w : [pre, post_1] \sqsubseteq w : [pre, post_2]$
    provided $pre[w \setminus w_0] \wedge post_2 \Rightarrow post_1$

Law $assignIV(w := E)$.
    $w, x : [pre, post] \sqsubseteq w := E$
    provided $(w = w_0) \wedge pre \Rightarrow post[w \setminus E]$

Law $skipIntroIV()$.
    $w : [pre, post] \sqsubseteq$ **skip**
    provided $(w = w_0) \wedge pre \Rightarrow post$

Law $contractFrame(x)$.
    $w, x : [pre, post] \sqsubseteq w : [pre, post[x_0 \setminus x]]$

Law $iter(\langle G_1, ..., G_n \rangle, V)$. For any formulae inv, the invariant; and any
 integer expression V, the variant, $w : [pre, post] \sqsubseteq$
        do $(?\ i.G_i \rightarrow w : [inv \wedge G_i, inv \wedge 0 \leq V \leq V[w \setminus w_0]])$ **od**

Law $varInt(n : T)$.
    $w : [pre, post] \sqsubseteq [\![\textbf{var}\ x : T \bullet w, x : [pre, post]]\!]$
    provided x does not occurs in w, pre, and $post$.

Law $conInt(c : T, pre')$.
    $w : [pre, post] \sqsubseteq [\![\textbf{con}\ c : T \bullet w : [pre', post]]\!]$
    provided $pre \Rightarrow (\exists\, c : T \bullet pre')$ and c does not occurs in w, pre and $post$.

Law $fixInitValue(c : T, E)$.
    For any term E
    $w : [pre, post] \sqsubseteq [\![\textbf{con}\ c : T \bullet w : [pre \wedge c = E, post]]\!]$
    provided $pre \Rightarrow E \in T$ and c is a new name.

Law $removeCon(x)$.
    $[\![\textbf{con}\ c : T \bullet p]\!] \sqsubseteq p$
    provided c does not occur in p.

Law $expandFrame(x)$.
    $w : [pre, post] \sqsubseteq w, x : [pre, post \wedge x = x_0]$

Law $seqCompCon(mid, X, x)$.
 $w, x : [pre, post] \sqsubseteq$
  $[\![\, \mathbf{con}\ X \bullet x : [pre, mid];\ w, x : [mid[x_0 \setminus X], post[x_0 \setminus X]]$
 provided mid does not contain other initial variables beyond x_0.

Law $seqCompIV(mid, x)$.
 $w, x : [pre, post] \sqsubseteq x : [pre, mid]\ ;\ w, x : [mid, post]$
 provided mid does not contain any initial variables and $post$ does not
 contain x_0.

Law $procNoArgsIntro(pn, p_1)$. $p_2 = [\![\, \mathbf{proc}\ pn = p_1 \bullet p_2]\!]$
 provided pn is not free in p_2

Law $procNoArgsCall()$. $[\![\, \mathbf{proc}\ pn = p_1 \bullet p_2[p_1]]\!] = [\![\, \mathbf{proc}\ pn = p_1 \bullet p_2[pn]]\!]$

Law $procArgsIntro(pn, p_1, par)$. $p_2 = [\![\, \mathbf{proc}\ pn = (par \bullet p_1) \bullet p_2]\!]$
 provided pn is not free in p_2

Law $procArgsCall()$.
 $[\![\, \mathbf{proc}\ pn =$
  $(par \bullet p_1) \bullet p_2[(par \bullet p_1)(a)]]\!] = [\![\, \mathbf{proc}\ pn = (par \bullet p_1) \bullet p_2[pn(a)]]\!]$

Law $callByValue(f, a)$.
 $w : [pre[f \setminus a], post[f, f_0 \setminus a, a_0]] =$
  $(\mathbf{val}\ f \bullet w : [pre, post])(a)$
 provided f is not in w and w is not free in a

Law $callByValueIV(f, a)$.
 $w : [pre[f \setminus a], post[f_0 \setminus a_0]] =$
  $(\mathbf{val}\ f \bullet w, f : [pre, post])(a)$
 provided f is not in w and w is not free in $post$

Law $callByResult(f, a)$. $w, a : [pre, post] = (\mathbf{res}\ f \bullet w, f : [pre, post[a \setminus f]])(a)$
 provided f is not in w, and is not free in pre or $post$ and, f_0 is not free in
 $post$

Law $multiArgs()$. $(\mathbf{par_1}\ f_1 \bullet (\mathbf{par_2}\ f_2)(a_2))(a_1) = (\mathbf{par_1}\ f_1;\ \mathbf{par_2}\ f_2)(a_1, a_2)$

Law $callByValueResult(f, a)$.
 $w, a : [pre[f \setminus a], post[f_0 \setminus a_0]] = (\mathbf{val\text{-}res}\ f \bullet w, f : [pre, post[a \setminus f]])(a)$
 provided f is not in w, and is not free in $post$

Law $variantIntro(pr, p_1, n, e, pars)$.
 $p_2 = [\![\, \mathbf{proc}\ pr = (par \bullet w : [n = e \land pre, post])\ \mathbf{variant}\ n\ \mathbf{is}\ e \bullet\ p_2]\!]$
 provided pr and n are not free in e and p_2

Law *procVariantBlockCall*().
$$\llbracket\, \textbf{proc}\ \ pr = (par \bullet p_1)\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet p_2[(par \bullet p_3)(a)]\,\rrbracket$$
$$= \llbracket\, \textbf{proc}\ \ pr = (par \bullet p_1)\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet p_2[(pr)(a)]\,\rrbracket$$
provided pr is not recursive, n is not free in e and p_3, and $\{n = e\}p_3 \sqsubseteq p_1$

law *recursiveCall*().
$$\llbracket\, \textbf{proc}\ \ pr = (par \bullet p_1[(par \bullet \{0 \le e < n\}p_3)(a)])\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet p_2\,\rrbracket$$
$$= \llbracket\, \textbf{proc}\ \ pr = (par \bullet p_1[pr(a)])\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet p_2\,\rrbracket$$
provided n is not free in p_3 and
$$p_1[pr(a)],\ \text{and}\ \{n = e\}p_3 \sqsubseteq p_1[(par \bullet \{0 \le e < n\}p_3)(a)]$$

Law *variantNoArgsIntro*(pr, p_1, n, e).
$$p_2 = \llbracket\, \textbf{proc}\ pr = w : [n = e \wedge pre, post]\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet\ p_2\,\rrbracket$$
provided pr and n are not free in e and p_2

Law *procVariantBlockCallNoArgs*().
$$\llbracket\, \textbf{proc}\ pr = w : [n = e \wedge pre, post])\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet\ p_2[w : [pre, post]]\,\rrbracket$$
$$\sqsubseteq \llbracket\, \textbf{proc}\ pr = w : [n = e \wedge pre, post]\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet\ p_2[pr]\,\rrbracket$$
provided nm is not recursive, n is not free in e and $w : [pre, post]$.

Law *recursiveCallNoArgs*().
$$\llbracket\, \textbf{proc}\ pr = p_1[w : [0 \le e < n \wedge pre, post]]\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet\ p_2\,\rrbracket$$
$$\sqsubseteq \llbracket\, \textbf{proc}\ pr = p_1[pr(a)]\ \textbf{variant}\ \ n\ \textbf{is}\ e \bullet\ p_2\,\rrbracket$$
provided n is not free in $w : [pre, post]$ and $p_1[pr]$ and
$$w : [n = e \wedge pre, post] \sqsubseteq p_1.$$

Law *coercion*(). $[post] = : [true, post]$
provided *post* does not contain any initial variables.

Law *absCoercion*(). $w : [pre, post]; [post'] = w : [pre, post \wedge post']$

Law *intCoercion*(). $\textbf{skip} \sqsubseteq [post]$

Appendix D

Proofs of the Laws

In this Appendix we prove the laws presented in this thesis. The laws marked with * are not valid with infinite lists and are proved only for finite lists. The lemmas used are presented and proved in the Appendix D.6.

D.1   Basic properties of composition

Law $1(a)$.   **skip** ; $t = t$

$$[\![\mathbf{skip};\ t]\!]\Gamma_L\ \Gamma_T\ r$$

$$= \stackrel{\propto}{}\!\!/ \bullet ([\![t]\!]\Gamma_L\ \Gamma_T) * \bullet [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r \qquad\qquad [\text{Definition of ; }]$$

$$= \stackrel{\propto}{}\!\!/(([\![t]\!]\Gamma_L\ \Gamma_T) * ([\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r)) \qquad\qquad [\text{Functional Composition}]$$

$$= \stackrel{\propto}{}\!\!/(([\![t]\!]\Gamma_L\ \Gamma_T) * [r]) \qquad\qquad\qquad [\text{Definition of }\mathbf{skip}]$$

$$= \stackrel{\propto}{}\!\!/[[\![t]\!]\Gamma_L\ \Gamma_T\ r] \qquad\qquad\qquad\qquad [\text{Definition of } *]$$

$$= [\![t]\!]\Gamma_L\ \Gamma_T\ r \qquad\qquad\qquad\qquad\qquad [\text{Property of } \stackrel{\propto}{}\!\!/]$$

$\square$

Law $1(b)$. $t = \mathbf{skip};\ t$

$$[\![\mathbf{skip};\ t]\!]\Gamma_L\ \Gamma_T\ r$$

$$= \stackrel{\propto}{}\!\!/ \bullet ([\![t]\!]\Gamma_L\ \Gamma_T) * \bullet [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r \qquad\qquad [\text{Definition of ; }]$$

$$= \stackrel{\propto}{}\!\!/([\![t]\!]\Gamma_L\ \Gamma_T) * (([\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r)) \qquad\qquad [\text{Functional Composition}]$$

$$= \stackrel{\propto}{}\!\!/([\![t]\!]\Gamma_L\ \Gamma_T) * [r] \qquad\qquad\qquad [\text{Definition of }\mathbf{skip}]$$

$$= \stackrel{\propto}{}\!\!/[[\![t]\!]\Gamma_L\ \Gamma_T\ r] \qquad\qquad\qquad\qquad [\text{Definition of } *]$$

$$= [\![t]\!]\Gamma_L\ \Gamma_T\ r \qquad\qquad\qquad\qquad\qquad [\text{Property of } \stackrel{\propto}{}\!\!/]$$

$\square$

Law $2(a)$. $t \mid \mathbf{fail} = t$

$\llbracket t \mid \mathbf{fail} \rrbracket \Gamma_L\ \Gamma_T\ r$

$= (^{\bowtie}\!/[(\llbracket t \rrbracket \Gamma_L\ \Gamma_T), (\llbracket \mathbf{fail} \rrbracket \Gamma_L\ \Gamma_T)]^{\circ})\ r$ — [Definition of $\mid$]

$= {}^{\bowtie}\!/[\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r, \llbracket \mathbf{fail} \rrbracket \Gamma_L\ \Gamma_T\ r]$ — [Definition of $^{\circ}$] [Functional Composition]

$= {}^{\bowtie}\!/[\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r, [\,]]$ — [Definition of $\mathbf{fail}$]

$= \bigsqcup_{\infty}\{c : [\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r, [\,]] \bullet {}^{\infty}_{\wedge}\!/\ c\}$ — [Definition of $^{\bowtie}\!/$]

$= \bigsqcup_{\infty}\{^{\infty}_{\wedge}\!/(f, <\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r, [\,]>)\}$ — [Sets theory]

$= \bigsqcup_{\infty}\{^{\infty}_{\wedge}\!/(f, <\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r> {}^{\bowtie}\ ^{\infty}_{\wedge}\!/(f, <[\,]>))\}$ — [Definition of $^{\bowtie}\!/$]

$= \bigsqcup_{\infty}\{\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r\ ^{\bowtie}\ [\,]\}$ — [Definition of $^{\bowtie}\!/$]

$= \bigsqcup_{\infty}\{\{x : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r;\ y : [\,] \bullet x \wedge y\}\}$ — [Definition of $^{\bowtie}$]

$= \bigsqcup_{\infty}\{\{x : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet x \wedge (f, <>)\}\}$ — [Definition of $[\,]$]

$= \bigsqcup_{\infty}\{\{(s,l) : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet (s,l) \frown\ <>)\}\}$ — [Definition of $\wedge$]

$= \bigsqcup_{\infty}\{\{(s,l) : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet (s,l)\}\}$ — [Definition of $\frown$]

$= \bigsqcup_{\infty}\{\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r\}$ — [Set theory]

$= \bigcup(\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r)$ — [Definition of $\bigsqcup_{\infty}$]

$= \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r$ — [Definition of $\bigcup$]

$\square$

Law $2(b)$. $t = \mathbf{fail} \mid t$

$(\llbracket t \rrbracket \Gamma_L\ \Gamma_T)\ r$

$= \bigcup(\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r)$ — [Definition of $\bigcup$]

$= \bigsqcup_{\infty}\{\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r\}$ — [Definition of $\bigsqcup_{\infty}$]

$= \bigsqcup_{\infty}\{\{(s,l) : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet (s,l)\}\}$ — [Set theory]

$= \bigsqcup_{\infty}\{\{(s,l) : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet <>\ ^{\bowtie} (s,l))\}\}$ — [Definition of cat]

$= \bigsqcup_{\infty}\{\{x : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r \bullet (f, <>) \wedge x\}\}$ — [Definition of $\wedge$]

$= \bigsqcup_{\infty}\{\{x : \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r;\ y : [\,] \bullet y \wedge x\}\}$ — [Definition of $[\,]$]

$= \bigsqcup_{\infty}\{[\,]\ ^{\bowtie} \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r\}$ — [Definition of $\wedge$]

$= \bigsqcup_{\infty}\{^{\infty}_{\wedge}\!/(f, <[\,]>)\ ^{\bowtie}\ ^{\infty}_{\wedge}\!/(f, <\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r>)\}$ — [Definition of $^{\bowtie}\!/$]

$= \bigsqcup_{\infty}\{^{\infty}_{\wedge}\!/(f, <[\,]\ ^{\bowtie} \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r)\}$ — [Definition of $^{\bowtie}\!/$]

$= \bigsqcup_{\infty}\{c : [[\,], \llbracket t \rrbracket \Gamma_L\ \Gamma_T\ r] \bullet {}^{\infty}_{\wedge}\!/\ c\}$ — [Set theory]

$= {}^{\bowtie}\!/[[\,], (\llbracket t \rrbracket \Gamma_L\ \Gamma_T)\ r]$ — [Definition of $^{\bowtie}\!/$]

$= {}^{\bowtie}\!/[(\llbracket \mathbf{fail} \rrbracket \Gamma_L\ \Gamma_T)\ r, (\llbracket t \rrbracket \Gamma_L\ \Gamma_T)\ r]$ — [Definition of $\mathbf{fail}$]

$= {}^{\bowtie}\!/[(\llbracket \mathbf{fail} \rrbracket \Gamma_L\ \Gamma_T), (\llbracket t \rrbracket \Gamma_L\ \Gamma_T)]^{\circ}\ r$ — [Definition of $\circ$]

$= \llbracket \mathbf{fail} \mid t \rrbracket \Gamma_L\ \Gamma_T\ r$ — [Definition of $\mid$]

$\square$

112

Law $3(a)^*$. $\quad t;\ \mathbf{fail} = \mathbf{fail}$

$[\![t;\ \mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$
$= \bowtie/ \bullet ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of ;]
$= \bowtie/([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T) * ([\![t]\!]\Gamma_L\ \Gamma_T\ r)$ [Functional Composition]
$= \bowtie/[\![\ ]\!]^{length([\![t]\!]\Gamma_L\ \Gamma_T\ r)}$ [Definition of $\mathbf{fail}$ and $*$]

    where

$$[lst]^0 = [\ ]$$
$$[lst]^n = [lst] \bowtie ([lst]^{n-1})$$

    As $([\![t]\!]\Gamma_L\ \Gamma_T)\ r$ is a finite list, we follow as

$= [\ ]$ [Definition of $\bowtie/$]
$= ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of $\mathbf{fail}$]

                                              $\square$

Law $3(b)$. $\quad \mathbf{fail} = t;\ \mathbf{fail}$

$[\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r = [\ ]$ [Definition of $\mathbf{fail}$]
$= \bowtie/[\ ]$ [Property of $\bowtie/$]
$= \bowtie/([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T) * [\ ]$ [Definition of $*$]
$= \bowtie/ \bullet ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of $\mathbf{fail}$]
$= [\![t;\ \mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]

                                              $\square$

Law 4. $\quad t_1 \mid (t_2 \mid t_3) = (t_1 \mid t_2) \mid t_3$

$[\![t_1 \mid (t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r$
$= \bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T, [\![(t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\mid$]
                                                       [Functional Composition]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![(t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r)$ [Lemma D.6.17]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie (\bowtie/([\![t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T]^\circ\ r))$ [Definition of $\mid$]
                                                       [Functional Composition]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie (([\![t_2]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r))$ [Lemma D.6.17]
$= (([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)) \bowtie ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Lemma D.6.24]
$= (\bowtie/[([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![t_2]\!]\Gamma_L\ \Gamma_T)]^\circ\ r) \bowtie ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\bowtie/$ and $^\circ$]
$= ([\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\mid$]
                                                       [Functional Composition]
$= \bowtie/[[\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\bowtie/$ and $^\circ$]
$= [\![(t_1 \mid t_2) \mid t_3]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of $\mid$]

                                               $\square$

Law 5. $t_1;\ (t_2;\ t_3) = (t_1;\ t_2);\ t_3$

$(\llbracket t_1;\ (t_2;\ t_3) \rrbracket \Gamma_L\ \Gamma_T))$
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket (t_1;\ t_2) \rrbracket \Gamma_L\ \Gamma_T)$ [Definition of ;]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\bowtie\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T))$ [Definition of ;]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \bowtie\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)$ [Property of $\bullet$]
$= \bowtie\!/ \bullet \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * * \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)$ [Lemma D.6.2]
$= \bowtie\!/ \bullet \bowtie\!/ * \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * * \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)$ [Lemma D.6.3]
$= \bowtie\!/ \bullet (\bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)$ [Property of $\bullet$ and $*$]
$= \bowtie\!/ \bullet ((\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T);\ (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T)) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)$ [Definition of ;]
$= (\llbracket t_1;\ (t_2;\ t_3) \rrbracket \Gamma_L\ \Gamma_T)$ [Definition of ;]

$\square$

Law 6. $(t_1 \mid t_2);\ t_3 = (t_1;\ t_3) \mid (t_2;\ t_3)$

$\llbracket (t_1 \mid t_2);\ t_3 \rrbracket \Gamma_L\ \Gamma_T\ r$
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket (t_1 \mid t_2) \rrbracket \Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \bowtie\!/ \bullet [\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\mid\mid$]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \bowtie\!/[\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r]$ [Definition of $^\circ$]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) \bowtie (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)$ [Definition of $\bowtie\!/$]
$= (\bowtie\!/ \bullet ((\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r)) \bowtie (\bowtie\!/ \bullet ((\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)\ r)$
      [Lemma D.6.19]
$= (\llbracket (t_1;\ t_3) \rrbracket \Gamma_L\ \Gamma_T)\ r \bowtie (\llbracket (t_2;\ t_3) \rrbracket \Gamma_L\ \Gamma_T)\ r$ [Definition of ;]
$= \bowtie\!/ \bullet [\llbracket (t_1;\ t_3) \rrbracket \Gamma_L\ \Gamma_T, \llbracket (t_2;\ t_3) \rrbracket \Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\bowtie\!/$ and $^\circ$]
$= \llbracket (t_1;\ t_3) \mid (t_2;\ t_3) \rrbracket \Gamma_L\ \Gamma_T\ r$ [Definition of $\mid\mid$]

$\square$

Law 6′. $(\mid_{i:I} t_i);\ t_3 = (\mid_{i:I} t_i;\ t_3)$

$\llbracket (\mid_{i:I} t_i);\ t_3 \rrbracket \Gamma_L\ \Gamma_T\ r$
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket (\mid_{i:I} t_i) \rrbracket \Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \bowtie\!/ \bullet [\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T, \ldots, \llbracket t_n \rrbracket \Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\mid\mid$]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \bowtie\!/[\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r, \ldots, \llbracket t_n \rrbracket \Gamma_L\ \Gamma_T\ r]$ [Definition of $^\circ$]
$= \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) \bowtie \ldots \bowtie (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)$ [Definition of $\bowtie\!/$]
$= (\bowtie\!/ \bullet ((\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r)) \bowtie \ldots \bowtie (\bowtie\!/ \bullet ((\llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T) * \bullet \llbracket t_n \rrbracket \Gamma_L\ \Gamma_T)\ r)$
      [Lemma D.6.19]
$= (\llbracket (t_1;\ t_3) \rrbracket \Gamma_L\ \Gamma_T)\ r \bowtie \ldots \bowtie (\llbracket (t_n;\ t_3) \rrbracket \Gamma_L\ \Gamma_T)\ r$ [Definition of ;]
$= \bowtie\!/ \bullet [\llbracket (t_1;\ t_3) \rrbracket \Gamma_L\ \Gamma_T, \ldots, \llbracket (t_n;\ t_3) \rrbracket \Gamma_L\ \Gamma_T]^\circ\ r$ [Definition of $\bowtie\!/$ and $^\circ$]
$= \llbracket (\mid_{i:I} t_i;\ t_3) \rrbracket \Gamma_L\ \Gamma_T\ r$ [Definition of $\mid\mid$]

$\square$

Law 7. $t_1; \, (t_2 \mid t_3) = (t_1; \, t_2) \mid (t_1; \, t_3)$ for any sequential tactic.

$$\llbracket t_1; \, (t_2 \mid t_3)\rrbracket \Gamma_L \, \Gamma_T \, r$$
$$= \bowtie\!/ \bullet (\llbracket t_2 \mid t_3 \rrbracket \Gamma_L \, \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r) \qquad [\text{Definition of } ; \,]$$
$$= \bowtie\!/ \bullet (\bowtie\!/ \bullet \llbracket \llbracket t_2 \rrbracket \Gamma_L \, \Gamma_T, \llbracket t_3 \rrbracket \Gamma_L \, \Gamma_T]^\circ) * \bullet \llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } !]$$
$$= \bowtie\!/[\bowtie\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L \, \Gamma_T) * \bullet \llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r, \bowtie\!/ \bullet (\llbracket t_3 \rrbracket \Gamma_L \, \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r)]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Definition of } ^\circ \text{ and } *]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Property of } *]$$
$$= \bowtie\!/ \bullet \llbracket \llbracket t_1; \, t_2 \rrbracket \Gamma_L \, \Gamma_T \, r, \llbracket t_1; \, t_3 \rrbracket \Gamma_L \, \Gamma_T \, r] \qquad [\text{Definition of } ; \,]$$
$$= \bowtie\!/ \bullet \llbracket \llbracket t_1; \, t_2 \rrbracket \Gamma_L \, \Gamma_T, \llbracket t_1; \, t_3 \rrbracket \Gamma_L \, \Gamma_T]^\circ \, r \qquad [\text{Definition of } ^\circ]$$
$$= \llbracket (t_1; \, t_2) \mid (t_1; \, t_3)\rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } !]$$

$\square$

Law 7′. $t_1; \, (\mid_{i:I} t_i) = (\mid_{i:I} t_1; \, t_i)$ for any sequential tactic.

$$\llbracket t_1; \, (\mid_{i:I} t_i)\rrbracket \Gamma_L \, \Gamma_T \, r$$
$$= \bowtie\!/ \bullet (\llbracket (\mid_{i:I} t_i)\rrbracket \Gamma_L \, \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r) \qquad [\text{Definition of } ; \,]$$
$$= \bowtie\!/ \bullet (\bowtie\!/ \bullet \llbracket \llbracket t_2 \rrbracket \Gamma_L \, \Gamma_T, \dots, \llbracket t_n \rrbracket \Gamma_L \, \Gamma_T]^\circ) * \bullet \llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } !]$$
$$= \bowtie\!/[\bowtie\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L \, \Gamma_T) * \bullet \llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r, \dots, \bowtie\!/ \bullet (\llbracket t_n \rrbracket \Gamma_L \, \Gamma_T) * \bullet (\llbracket t_1 \rrbracket \Gamma_L \, \Gamma_T \, r)]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Definition of } ^\circ \text{ and } *]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Property of } *]$$
$$= \bowtie\!/ \bullet \llbracket \llbracket t_1; \, t_2 \rrbracket \Gamma_L \, \Gamma_T \, r, \dots, \llbracket t_1; \, t_n \rrbracket \Gamma_L \, \Gamma_T \, r] \qquad [\text{Definition of } ; \,]$$
$$= \bowtie\!/ \bullet \llbracket \llbracket t_1; \, t_2 \rrbracket \Gamma_L \, \Gamma_T, \dots, \llbracket t_1; \, t_n \rrbracket \Gamma_L \, \Gamma_T]^\circ \, r \qquad [\text{Definition of } ^\circ]$$
$$= \llbracket (\mid_{i:I} t_1; \, t_i)\rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } !]$$

$\square$

D.2   Laws involving Cut

Law 8.   !**skip** = **skip**

$$\llbracket !\mathbf{skip}\rrbracket \Gamma_L \, \Gamma_T \, r$$
$$= head' \bullet \llbracket \mathbf{skip}\rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } !]$$
$$= head'[r] \qquad [\text{Definition of } \mathbf{skip}]$$
$$= [r] \qquad [\text{Definition of } head']$$
$$= \llbracket \mathbf{skip}\rrbracket \Gamma_L \, \Gamma_T \, r \qquad [\text{Definition of } \mathbf{skip}]$$

$\square$

115

Law 9.   !**fail** = **fail**

$[\![!\textbf{fail}]\!]\Gamma_L \; \Gamma_T \; r$
$= head' \bullet [\![\textbf{fail}]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of !]
$= head'[\,]$ [Definition of **fail**]
$= [\,]$ [Definition of $head'$]
$= [\![\textbf{fail}]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of **fail**]

$\square$

Law 10.   !(**law** $l \; (args)$) = **law** $l \; (args)$

$[\![! \, \textbf{law} \; l \; (args)]\!]\Gamma_L \; \Gamma_T \; r$
$= head' \bullet ([\![\, \textbf{law} \; l \; (args)]\!]\Gamma_L \; \Gamma_T \; r)$ [Definition of !]
$= head'([\![\, \textbf{law} \; l \; (args)]\!]\Gamma_L \; \Gamma_T \; r)$ [Functional Composition]
$= [\![\, \textbf{law} \; l \; (args)]\!]\Gamma_L \; \Gamma_T \; r$ [Lemma D.6.20]
             [Definition of $head'$]

$\square$

Law 11. $!t_1; \; (t_2 \mid t_3) = (!t_1; \; t_2) \mid (!t_1; \; t_3)$

$[\![!t_1; \; (t_2 \mid t_3)]\!]\Gamma_L \; \Gamma_T \; r$
$= ([\![(t_2 \mid t_3)]\!]\Gamma_L \; \Gamma_T)* \bullet [\![!t_1]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of ;]
$= (\Join\!/ \bullet [\![t_2]\!]\Gamma_L \; \Gamma_T, [\![t_3]\!]\Gamma_L \; \Gamma_T]^\circ)* \bullet [\![!t_1]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of $\mid\mid$]
$= (\Join\!/ \bullet ([\![t_2]\!]\Gamma_L \; \Gamma_T) * ([\![!t_1]\!]\Gamma_L \; \Gamma_T \; r) \Join ([\![t_3]\!]\Gamma_L \; \Gamma_T) * ([\![!t_1]\!]\Gamma_L \; \Gamma_T \; r)$
             [Definition of $\circ$ and $*$]
$= (\Join\!/ \bullet ([\![t_2]\!]\Gamma_L \; \Gamma_T) * ([\![!t_1]\!]\Gamma_L \; \Gamma_T \; r)) \Join (\Join\!/ \bullet ([\![t_3]\!]\Gamma_L \; \Gamma_T) * ([\![!t_1]\!]\Gamma_L \; \Gamma_T \; r))$
             [Lemma D.6.5]
$= ([\![!t_1; \; t_2]\!]\Gamma_L \; \Gamma_T) \Join ([\![!t_1; \; t_3]\!]\Gamma_L \; \Gamma_T \; r)$ [Definition of ;]
$= \Join\!/ \bullet [\![!t_1; \; t_2]\!]\Gamma_L \; \Gamma_T, [\![!t_1; \; t_3]\!]\Gamma_L \; \Gamma_T]^\circ \; r$ [Definition of $\Join\!/$ and $\circ$]
$= [\![(!t_1; \; t_2) \mid (!t_1; \; t_3)]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of $\mid\mid$]

$\square$

Law 11′. $!t_1; \; (\mid_{i:I} t_i) = \mid_{i:i} t_1; \; t_i$

$[\![!t_1; \; (\mid_{i:I} t_i)]\!]\Gamma_L \; \Gamma_T \; r$
$= ([\![(\mid_{i:I} t_i)]\!]\Gamma_L \; \Gamma_T)* \bullet [\![!t_1]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of ;]
$= (\Join\!/ \bullet [\![t_2]\!]\Gamma_L \; \Gamma_T, \ldots, [\![t_n]\!]\Gamma_L \; \Gamma_T]^\circ)* \bullet [\![!t_1]\!]\Gamma_L \; \Gamma_T \; r$ [Definition of $\mid\mid$]

$= (\bowtie/ \bullet (\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) * (\llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r) \bowtie \ldots \bowtie (\llbracket t_n \rrbracket \Gamma_L \ \Gamma_T) * (\llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)$
$\hfill$ [Definition of $\circ$ and $*$]
$= (\bowtie/ \bullet (\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) * (\llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)) \bowtie \ldots \bowtie (\bowtie/ \bullet (\llbracket t_n \rrbracket \Gamma_L \ \Gamma_T) * (\llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))$
$\hfill$ [Lemma D.6.5]
$= (\llbracket !t_1; \ t_2 \rrbracket \Gamma_L \ \Gamma_T) \bowtie \ldots \bowtie (\llbracket !t_1; \ t_n \rrbracket \Gamma_L \ \Gamma_T \ r)$ $\hfill$ [Definition of ;]
$= \bowtie/ \bullet [\llbracket !t_1; \ t_2 \rrbracket \Gamma_L \ \Gamma_T, \ldots, \llbracket !t_1; \ t_n \rrbracket \Gamma_L \ \Gamma_T]^{\circ} \ r$ $\hfill$ [Definition of $\bowtie/$ and $\circ$]
$= \ |\ _{i:i} \ t_1; \ t_i \rrbracket \Gamma_L \ \Gamma_T \ r$ $\hfill$ [Definition of $|$]

$\hfill \square$

Law 12. $\ !t_1; \ !t_2 = !(!t_1; \ !t_2)$

$\llbracket !t_1; \ !t_2 \rrbracket \Gamma_L \ \Gamma_T \ r$
$= \bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * \bullet \ \llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r$ $\hfill$ [Definition of ;]
$= \bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * \bullet \ head'(\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)$ $\hfill$ [Definition of !]
$= \bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * [head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)]$ $\hfill$ [Definition of $head'$]
$= \bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)$ $\hfill$ [Definition of $*$]
$= \bowtie/ \bullet head'(((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))$ $\hfill$ [Definition of !]
$= \bowtie/ \bullet [head \ (((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))]$ $\hfill$ [Definition of $head'$]
$= [head \ (((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))]$ $\hfill$ [Definition of $\bowtie/$]
$= [head \ ([head \ (((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))])]$ $\hfill$ [Definition of $take$]
$= head'([head \ (((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))])$ $\hfill$ [Definition of $head'$]
$= head'(\bowtie/[head \ (((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))])$ $\hfill$ [Definition of $\bowtie/$]
$= head'(\bowtie/ \bullet head'(((\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)))$ $\hfill$ [Definition of $head'$]
$= head'(\bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) \ head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))$ $\hfill$ [Definition of !]
$= head'(\bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * \bullet [head \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)])$ $\hfill$ [Definition of $*$]
$= head'(\bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * \bullet head'((\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r))$ $\hfill$ [Definition of $head'$]
$= head'(\bowtie/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L \ \Gamma_T) * \bullet (\llbracket !t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)$ $\hfill$ [Definition of !]
$= head'(\llbracket (!t_1; \ !t_2) \rrbracket \Gamma_L \ \Gamma_T \ r)$ $\hfill$ [Definition of ;]
$= \llbracket !(!t_1; \ !t_2) \rrbracket \Gamma_L \ \Gamma_T \ r$ $\hfill$ [Definition of !]

$\hfill \square$

Law 13. $!(t_1;\ t_2) = !(t_1;\ !t_2)$

$[\![!(t_1;\ t_2)]\!]\Gamma_L\ \Gamma_T\ r$

$= head'([\![t_1;\ t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]

$= head'(\bowtie\!/\ \bullet\ ([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of ;]

$= [head(\bowtie\!/\ \bullet\ ([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)]$ [Definition of $head'$]

$=]head([head((([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)])]$ [Definition of $\bowtie\!/$]

$= head'([head((([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)])$ [Definition of $head'$]

$= head'(\bowtie\!/\ \bullet\ [head((([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)])$ [Definition of $\bowtie\!/$]

$= head'(\bowtie\!/\ \bullet\ head'\ \bullet\ ([\![t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $head'$]

$= head'(\bowtie\!/\ \bullet\ ([\![!t_2]\!]\Gamma_L\ \Gamma_T)*\ \bullet\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]

$= head'([\![t_1;\ !t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of ;]

$= [\![!(t_1;\ !t_2)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]

$\square$

Law 14(a). $!(t_1\ |\ t_2) = !(!t_1\ |\ t_2)$

$[\![!(t_1\ |\ t_2)]\!]\Gamma_L\ \Gamma_T\ r$

$= head'([\![t_1\ |\ t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]

$= head'(\bowtie\!/\ \bullet\ [[\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T]^\circ\ r)$ [Definition of !]

$= head'(\bowtie\!/[[\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r]$ [Definition of $^\circ$]

$= head'((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)\ \bowtie\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\bowtie\!/$]

$= [head\ ((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)\ \bowtie\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))]$ [Definition of $head'$]

   Case $[\![t_1]\!]\Gamma_L\ \Gamma_T\ r = [\ ]$

$= [head\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)]$ [Definition of $\bowtie$]

$= [head\ ((head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r))\ \bowtie\ [\![t_2]\!]\Gamma_L\ \Gamma_T\ r)]$ [Definition of $head', \bowtie$]

                 [Assumption]

$= head'(\bowtie\!/[head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r), [\![t_2]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of $\bowtie\!/$]

$= head'(\bowtie\!/[head'([\![t_1]\!]\Gamma_L\ \Gamma_T), [\![t_2]\!]\Gamma_L\ \Gamma_T]^\circ\ r)$ [Definition of $^\circ$]

$= [\![!(!t_1\ |\ t_2)]\!]\Gamma_L\ \Gamma_T$ [Definition of !, ||]

   Case $[\![t_1]\!]\Gamma_L\ \Gamma_T\ r \neq [\ ]$

$= [head\ ([head\ ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)]\ \bowtie\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))]$ [Lemma D.6.22]

$= head'([head\ ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)]\ \bowtie\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $head'$]

$= head'(\bowtie\!/[[head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)], [\![t_2]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of $\bowtie\!/$]

$= head'(\bowtie\!/[head'([\![t_1]\!]\Gamma_L\ \Gamma_T)\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of $head'$]

$= head'(\bowtie\!/[[\![!t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of !]

$= head'(\bowtie\!/\ \bullet\ [[\![!t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T]^\circ\ r)$ [Definition of $^\circ$]

$= head'([\![!t_1\ |\ t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]

$= [\![!(!t_1\ |\ t_2)]\!]\Gamma_L\ \Gamma_T$ [Definition of !]

$\square$

118

Law 14(b). $!(t_1 \mid t_2) =!(t_1 \mid !t_2)$

$[\![!(t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T\ r$
$= head'([\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= head'(\bowtie/ \bullet [[\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T]^\circ\ r)$ [Definition of !]
$= head'(\bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r]$ [Definition of $^\circ$]
$= head'(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\bowtie/$]
$= [head((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))]$ [Definition of $head'$]

    Case $[\![t_2]\!]\Gamma_L\ \Gamma_T\ r = [\ ]$

$= [head\ ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)]$ [Definition of $\bowtie$]
$= [head\ ((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie head'([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))]$ [Definition of $head', \bowtie$]
                                                         [Assumption]
$= head'(\bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T\ r, head'([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)])$ [Definition of $\bowtie/$]
$= head'(\bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T, head'([\![t_2]\!]\Gamma_L\ \Gamma_T)]^\circ\ r)$ [Definition of $^\circ$]
$= [\![!(!t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T$ [Definition of !, |]

    Case $[\![t_2]\!]\Gamma_L\ \Gamma_T\ r \neq [\ ]$

$= [head((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie [head\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)])]$ [Lemma D.6.23]
$= head'((([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \bowtie [head\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)])$ [Definition of $head'$]
$= head'(\bowtie/[(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r), [head\ ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)]])$ [Definition of $\bowtie/$]
$= head'(\bowtie/[(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r), head'([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)])$ [Definition of $head'$]
$= head'(\bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![!t_2]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of !]
$= head'(\bowtie/ \bullet [([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![!t_2]\!]\Gamma_L\ \Gamma_T)]^\circ\ r)$ [Definition of $^\circ$]
$= head'([\![t_1 \mid !t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= [\![!(t_1 \mid !t_2)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]

                                                                        □

Law 15. $!(t_1 \mid t_1;\ t_2) =!t_1$

$[\![!(t_1 \mid t_1;\ t_2)]\!]\Gamma_L\ \Gamma_T\ r$
$= head'([\![t_1 \mid t_1;\ t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= head'(\bowtie/ \bullet [[\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_1;\ t_2]\!]\Gamma_L\ \Gamma_T]^\circ\ r)$ [Definition of |]
$= head'(\bowtie/ \bullet [[\![t_1]\!]\Gamma_L\ \Gamma_T, \bowtie/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)]^\circ\ r)$

                                                                 [Definition of ;]
$= head'(\bowtie/[[\![t_1]\!]\Gamma_L\ \Gamma_T\ r, \bowtie/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r])$ [Definition of $^\circ$]
                                                            [Functional Composition]
$= head'((([\![t_1]\!]\Gamma_L\ \Gamma_T)\ r \bowtie (\bowtie/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\bowtie/$]
$= head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Lemma D.6.6]
$= [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]

                                                                        □

Law 16. $!(t_1 \mid t_2 \mid t_1) \, = \, !(t_1 \mid t_2)$

$\llbracket !(t_1 \mid t_2 \mid t_1) \rrbracket \Gamma_L \ \Gamma_T \ r$
$= head'(\llbracket (t_1 \mid t_2 \mid t_1) \rrbracket \Gamma_L \ \Gamma_T) \ r$ [Definition of !]
$= head'(\bowtie\!/ \bullet [\llbracket t_1 \mid t_2 \rrbracket \Gamma_L \ \Gamma_T, \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T]^\circ \ r)$ [Definition of ||]
$= head'(\bowtie\!/[(\bowtie\!/ \bullet [\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T, \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T]^\circ), (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T)]^\circ \ r)$

[Definition of ||]

$= head'(\bowtie\!/[(\bowtie\!/[(\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T), (\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T)]^\circ) \ r, \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r])$

[Definition of °]

$= head'(\bowtie\!/[(\bowtie\!/ \bullet [\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r, \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ r]), \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r])$

[Definition of °]
$= head'(\bowtie\!/ \bullet [\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r \ \bowtie \ \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ r, \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r])$ [Definition of $\bowtie\!/$]
$= head'(\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r \ \bowtie \ \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ r \ \bowtie \ \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of $\bowtie\!/$]
$= head'(\bowtie\!/[\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ r, \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ r])$ [Lemma D.6.7]
$= head'(\bowtie\!/[\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T, \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T]^\circ \ r)$ [Definition of °]
$= head'(\llbracket t_1 \mid t_2 \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of ||]
$= \llbracket !(t_1 \mid t_2) \rrbracket \Gamma_L \ \Gamma_T \ r$ [Definition of !]

$\square$

Law 17. $!(\mathbf{skip} \mid t) = \mathbf{skip}$

$\llbracket !(\mathbf{skip} \mid t) \rrbracket \Gamma_L \ \Gamma_T \ r$
$= head'(\llbracket (\mathbf{skip} \mid t) \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of !]
$= head'(\bowtie\!/ \bullet [\llbracket \mathbf{skip} \rrbracket \Gamma_L \ \Gamma_T, \llbracket t \rrbracket \Gamma_L \ \Gamma_T]^\circ \ r)$ [Definition of ||]
$= head'(\bowtie\!/[\llbracket \mathbf{skip} \rrbracket \Gamma_L \ \Gamma_T \ r, \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r])$ [Definition of °]

[Functional Composition]
$= head'(\bowtie\!/[[r], \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r])$ [Definition of **skip**]
$= head'([r] \ \bowtie \ \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of $\bowtie\!/$]
$= [head([r] \ \bowtie \ \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r])]$ [Definition of $head'$]
$= [r]$ [Definition of $head$]
$= \llbracket \mathbf{skip} \rrbracket \Gamma_L \ \Gamma_T \ r$ [Definition of **skip**]

$\square$

Law 18. $!(t \mid t) \, = \, !t$

$\llbracket !(t \mid t) \rrbracket \Gamma_L \ \Gamma_T \ r$
$= head'(\llbracket (t \mid t) \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of !]
$= head'(\bowtie\!/ \bullet [\llbracket t \rrbracket \Gamma_L \ \Gamma_T, \llbracket t \rrbracket \Gamma_L \ \Gamma_T]^\circ \ r)$ [Definition of ||]
$= head'(\bowtie\!/[\llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r, \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r])$ [Definition of °]
$= head'(\llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r \ \bowtie \ \llbracket t \rrbracket \Gamma_L \ \Gamma_T \ r)$ [Definition of $\bowtie\!/$]

$= [head(\llbracket t \rrbracket \Gamma_L \; \Gamma_T \; r \; ⅋ \; \llbracket t \rrbracket \Gamma_L \; \Gamma_T \; r)]$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of $head'$]
$= [head(\llbracket t \rrbracket \Gamma_L \; \Gamma_T \; r)]$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Property of $head$]
$= head'(\llbracket t \rrbracket \Gamma_L \; \Gamma_T \; r)$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of $head'$]
$= \llbracket !t \rrbracket \Gamma_L \; \Gamma_T \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of !]

⠀⠀⠀ □

Law 19. ⠀ $!!t = !t$

$!!t \; r = !t \; r$
$= head'(head'(t \; r))$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of !]
$= head'(t \; r)$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Lemma D.6.15]
$= !t \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of !]

⠀⠀⠀ □

D.3 ⠀ Laws involving *succs* and *fails*

In this section we use the notation

⠀⠀⠀$t_1 \lhd condition \rhd t_2 = \textbf{if } condition \textbf{ then } t_1 \textbf{ else } t_2$

Law 20. ⠀ **succs** t; $t = t$

succs t; $t \; r$
$= ⅋ \bullet t * \bullet \textbf{succs} \; t \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of ;]

Case $t \; r = \bot$
$= ⅋/ \bullet t * \bot$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of **succs**]
$= \bot$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of **fail**]
$= t \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Assumption]

Case $t \; r = [\;]$
$= ⅋/ \bullet t * \bullet \textbf{fail} \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of **succs**]
$= ⅋/ \bullet t * [\;]$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of **fail**]
$= ⅋/ [\;]$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of $*$]
$= [\;]$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of $⅋/$]
$= t \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Assumption]

Case $t \; r \neq [\;]$
$= ⅋/ \bullet t * \bullet \textbf{skip} \; r$ ⠀⠀⠀⠀⠀⠀⠀⠀⠀ [Definition of **succs**]

121

$= \bowtie / \bullet\ t * [r]$ [Definition of **skip**]

$= \bowtie /[t\ r]$ [Definition of $*$]

$= t\ r$ [Definition of $\bowtie /$]

$\square$

Law 21. $*$ **fails** $t;\ t =$ **fail**

fails $t;\ t\ r$

$= \bowtie / \bullet\ t* \bullet$ **fails** $t\ r$ [Definition of ;]

Case $t\ r = <>$

$= \bowtie / \bullet\ t* \bullet$ **skip** r [Definition of **succs**]

$= \bowtie / \bullet\ t * [r]$ [Definition of **skip**]

$= \bowtie / \bullet\ [t\ r]$ [Definition of $*$]

$= t\ r$ [Definition of $\bowtie /$]

$= [\ ]$ [Assumption]

$=$ **fail** r [Definition of **fail**]

Case $t\ r \neq [\ ]$

$= \bowtie / \bullet\ t* \bullet$ **fail** r [Definition of **fails**]

$= \bowtie /[\ ]$ [Definition of $*$]

$=$ **fail** r [Definition of $\bowtie /$]

$\square$

Law 22.   **fails** $t =$ **fails** $!t =!$ **fails** t

1. **fails** $t =$ **fails** $!t$

fails $t\ r$

Case $t\ r = [\ ]$

$=$ **skip** r [Definition of **fails**]

$=$ **fails**$(head'(t\ r))$ [Definition of **fails**]

          [Definition of $head'$]

          [Assumption]

$=$ **fails**$(!t\ r)$ [Definition of !]

Case $t\ r \neq [\ ]$

$=$ **fail** r [Definition of **fails**]

$= \textbf{fails}(head'(t\ r))$ [Definition of **fails**]

[Definition of $head'$]

[Assumption]

$= \textbf{fails}(!t\ r)$ [Definition of !]

Case $t\ r = \bot$

$= \bot$ [Definition of **fails**]

$= \textbf{fails}(!t\ r)$ [Definition of !, $head'$]

[Definition of **fails**$'$]

2. **fails** $!t =!$ **fails** $t\ r$

fails $!t$

Case $t\ r = [\ ]$

$= (\textbf{skip} \lhd !t\ r = [\ ] \rhd \textbf{fail})\ r$ [Definition of **fails**]

$= (\textbf{skip} \lhd head'(t\ r) = [\ ] \rhd \textbf{fail})\ r$ [Definition of !]

$= (\textbf{skip} \lhd head'([\ ]) = [\ ] \rhd \textbf{fail})\ r$ [Assumption]

$= \textbf{skip}\ r$ [Definition of head']

$= !\textbf{skip}\ r$ [Law 8]

$= !(\textbf{fails}\ t\ r)$ [Definition of **fails**]

[Assumption]

Case $t\ r = lst \neq [\ ]$

$= (\textbf{skip} \lhd !t\ r = [\ ] \rhd \textbf{fail})\ r$ [Definition of **fails**]

$= (\textbf{skip} \lhd head'(lst) = [\ ] \rhd \textbf{fail})\ r$ [Definition of !]

[Assumption]

$= \textbf{fail}\ r$ [Definition of $\lhd\ \rhd$]

$= !\textbf{fail}\ r$ [Law 9]

$= !(\textbf{fails}\ t\ r)$ [Definition of **fails**]

[Assumption]

Case $t\ r = \bot$

$= \bot$ [Definition of **fails**]

[Definition of **succs**]

$= !(\textbf{fails}\ t\ r)$ [Definition of **fails**,!]

[Definition of $head'$]

$\square$

Law 23. **fails**(**succs** t) = **fails** t

fails(**succs** t) r

Case t $r = [\,]$
fails(**succs** t) r
$= ($**skip** $\lhd$ **succs** t $r = [\,] \rhd$ **fail**$)$ r [Definition of **fails**]
$= ($**skip** $\lhd$ **fail** $r = [\,] \rhd$ **fail**$)$ r [Definition of **succs**]
     [Assumption]
$= ($**skip** $\lhd$ $[\,] = [\,] \rhd$ **fail**$)$ r [Definition of **fail**]
$=$ **skip** r [Definition of $\lhd$ $\rhd$]
$=$ **fails** t r [Definition of **fails**]
     [Assumption]

Case t $r \neq [\,] = ($**skip** $\lhd$ **succs** t $r = [\,] \rhd$ **fail**$)$ r [Definition of **fails**]
$= ($**skip** $\lhd$ **skip** $r = [\,] \rhd$ **fail**$)$ r [Definition of **succs**]
     [Assumption]
$= ($**skip** $\lhd$ $[r] = [\,] \rhd$ **fail**$)$ r [Definition of **skip**]
$=$ **fail** r [Definition of $\lhd$ $\rhd$]
$=$ **fails** t r [Definition of **fails**]
     [Assumption]

Case t $r = \perp$
Case t $r = \perp$
$= \perp$ [Definition of **fails**
     [Definition of **succs**]
$=$ **fails** t [Assumption]
     [Definition of **fails**]

                       $\square$

Law 24*. **fails** t_1; **fails** t_2 = **fails** t_2; **fails** t_1

fails t_1; **fails** t_2 r
Case t_1 $r = [\,]$ and t_2 $r = [\,]$
$= \,^{\infty}\!\!/ \bullet ($**fails** $t_2)* \bullet$ **fails** t_1 r [Definition of ;]
$= \,^{\infty}\!\!/ \bullet ($**fails** $t_2)* \bullet$ **skip** r [Definition of **fails**]
$= \,^{\infty}\!\!/ \bullet ($**fails** $t_2) * [r]$ [Definition of **skip**]
$= \,^{\infty}\!\!/[$**fails** t_2 $r]$ [Definition of $*$]
$=$ **fails** t_2 r [Definition of $^{\infty}\!\!/$]
$=$ **skip** r [Definition of **fails**]
$=$ **fails** t_1 r [Definition of **fails**]
$= \,^{\infty}\!\!/[$**fails** t_1 $r]$ [Definition of $^{\infty}\!\!/$]

$$= \otimes\!/(\textbf{fails}\ t_1) * [r] \qquad\qquad\qquad \text{[Definition of } *]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{skip}\ r \qquad\qquad \text{[Definition of } \textbf{skip}]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fails}\ t_2\ r \qquad\quad \text{[Definition of } \textbf{fails}]$$
$$= \textbf{fails}\ t_2;\ \textbf{fails}\ t_1\ r \qquad\qquad\qquad \text{[Definition of ;]}$$

Case $t_1\ r = [\,]$ and $t_2\ r \neq [\,]$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{fails}\ t_1\ r \qquad \text{[Definition of ;]}$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{skip}\ r \qquad\ \text{[Definition of } \textbf{fails}]$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2) * [r] \qquad\qquad \text{[Definition of } \textbf{skip}]$$
$$= \otimes\!/[\textbf{fails}\ t_2\ r] \qquad\qquad\qquad\quad \text{[Definition of } *]$$
$$= \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\quad \text{[Definition of } \otimes\!/]$$
$$= \textbf{fail}\ r \qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } \textbf{fails}]$$
$$= [\,] \qquad\qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } fail]$$
$$= \otimes\!/[\,] \qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } \otimes\!/]$$
$$= \otimes\!/(\textbf{fails}\ t_1) * [\,] \qquad\qquad\quad \text{[Definition of } *]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fail}\ r \qquad\quad \text{[Definition of } \textbf{fail}]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fails}\ t_2\ r \quad \text{[Definition of } \textbf{fails}]$$
$$= \textbf{fails}\ t_2;\ \textbf{fails}\ t_1\ r \qquad\qquad\quad \text{[Definition of ;]}$$

Case $t_1\ r \neq [\,]$ and $t_2\ r = [\,]$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{fails}\ t_1\ r \qquad \text{[Definition of ;]}$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{fail}\ r \qquad\ \text{[Definition of } \textbf{fails}]$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2) * [\,] \qquad\qquad \text{[Definition of } \textbf{fail}]$$
$$= \otimes\!/[\,] \qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } *]$$
$$= \otimes\!/[[\,]] \qquad\qquad\qquad\qquad\quad \text{[Definition of } \otimes\!/]$$
$$= \otimes\!/[\textbf{fail}\ r] \qquad\qquad\qquad\qquad \text{[Definition of } \textbf{fail}]$$
$$= \otimes\!/[\textbf{fails}\ t_1\ r] \qquad\qquad\qquad \text{[Definition of } \textbf{fails}]$$
$$= \otimes\!/(\textbf{fails}\ t_1) * [r] \qquad\qquad\quad \text{[Definition of } *]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{skip}\ r \qquad\quad \text{[Definition of } \textbf{skip}]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fails}\ t_2\ r \quad \text{[Definition of } \textbf{fails}]$$
$$= \textbf{fails}\ t_2;\ \textbf{fails}\ t_1\ r \qquad\qquad\quad \text{[Definition of ;]}$$

Case $t_1\ r \neq [\,]$ and $t_2\ r \neq [\,]$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{fails}\ t_1\ r \qquad \text{[Definition of ;]}$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2)* \bullet \textbf{fail}\ r \qquad\ \text{[Definition of } \textbf{fails}]$$
$$= \otimes\!/ \bullet (\textbf{fails}\ t_2) * [\,] \qquad\qquad \text{[Definition of } \textbf{fail}]$$
$$= \otimes\!/[\,] \qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } *]$$
$$= [\,] \qquad\qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } \otimes\!/]$$
$$= \otimes\!/(\textbf{fails}\ t_1) * [\,] \qquad\qquad\quad \text{[Definition of } *, \otimes\!/]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fail}\ r \qquad\quad \text{[Definition of } \textbf{fail}]$$
$$= \otimes\!/(\textbf{fails}\ t_1)* \bullet \textbf{fails}\ t_2\ r \quad \text{[Definition of } \textbf{fails}]$$

$= \textbf{fails} \ t_2; \ \textbf{fails} \ t_1 \ r$ [Definition of ;]

$\square$

Law 25. $!(t_1 \mid t_2) = !t_1 \mid (\textbf{fails} \ t_1; \ !t_2)$

$!(t_1 \mid t_2) \ r$
$= head' \bullet {\bowtie}\!/ \bullet [t_1, t_2]^{\circ} \ r$ [Definition of !, |]
$= head' \bullet {\bowtie}\!/ \bullet [t_1 \ r, t_2 \ r]$ [Definition of $^{\circ}$]
$= head' \bullet {\bowtie}\!/[t_1 \ r, t_2 \ r]$ [Definition of $^{\circ}$]
$= head'(t_1 \ r \ {\bowtie} \ t_2 \ r)$ [Definition of ${\bowtie}\!/]$

Case $t_1 \ r = [\ ]$
$= head'(t_2 \ r)$ [Property of $head'$]
$= {\bowtie}\!/[[\ ], head'(t_2 \ r)]$ [Definition of ${\bowtie}\!/]$
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * [r]]$ [Definition of $head'$]
                                                [Definition of ${\bowtie}\!/, *$]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * \textbf{skip} \ r]$ [Definition of $\textbf{skip}$]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * \textbf{fails} \ t_1 \ r]$ [Definition of $\textbf{fails}$]
$= {\bowtie}\!/ \bullet [!t_1, \textbf{fails} \ t_1; \ !t_2]^{\circ} \ r$ [Definition of $^{\circ}, ; \ , !$]
$= !t_1 \mid (\textbf{fails} \ t_1; \ !t_2) \ r$ [Definition of |]

Case $t_1 \ r \neq [\ ] = head'(t_1 \ r \ {\bowtie} \ [\ ])$ [Assumption]
$= {\bowtie}\!/[head'(t_1 \ r), [\ ]]$ [Definition of ${\bowtie}\!/]$
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * [\ ])]$ [Definition of ${\bowtie}\!/, *$]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * \bullet \ \textbf{fails} \ t_1 \ r)]$ [Definition of $\textbf{fails}, \textbf{fail}$]
$= {\bowtie}\!/ \bullet [!t_1, \textbf{fails} \ t_1; \ !t_2)]^{\circ} \ r$ [Definition of !, ; \ , $^{\circ}$]
$= !t_1 \mid (\textbf{fails} \ t_1; \ !t_2) \ r$ [Definition of |]
Case $t_1 \ r = \perp$
$= \perp$ [Definition of ${\bowtie}, head'$]
$= {\bowtie}\!/[\perp, head'(t_2 \ r)]$ [Definition of ${\bowtie}\!/]$
$= {\bowtie}\!/[head'(\perp), head'(t_2 \ r)]$ [Definition of $head'$]
$= {\bowtie}\!/[head'(t_1 \ r), head'(t_2 \ r)]$ [Assumption]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * [r]]$ [Definition of $head'$]
                                                [Definition of ${\bowtie}\!/, *$]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * \textbf{skip} \ r]$ [Definition of $\textbf{skip}$]
$= {\bowtie}\!/[head'(t_1 \ r), {\bowtie}\!/ \bullet (head' \ t_2) * \textbf{fails} \ t_1 \ r]$ [Definition of $\textbf{fails}$]
$= {\bowtie}\!/ \bullet [!t_1, \textbf{fails} \ t_1; \ !t_2]^{\circ} \ r$ [Definition of $^{\circ}, ; \ , !$]
$= !t_1 \mid (\textbf{fails} \ t_1; \ !t_2) \ r$ [Definition of |]

$\square$

126

Law 26. $!(t_1;\ t_2) = !(t_1;\ \textbf{succs}\ t_2);\ !t_2$
  This Law is proved by induction as seen below

$!(t_1;\ t_2)\ r$
$= head' \bullet \bowtie/ \bullet t_2 * \bullet t_1\ r$ [Definition of !,;]

Base Case $t_1\ r = [\ ]$
$= head' \bullet \bowtie/ \bullet t_2 * [\ ]$ [Assumption]
$= head'\ [\ ]$ [Definition of $*,\bowtie/$]
$= [\ ]$ [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2) * [\ ]$ [Definition of $*,\bowtie/$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'\ [\ ]$ [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * [\ ])$ [Definition of $*,\bowtie/$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * \bullet t_1\ r)$ [Assumption]
$= !(t_1;\ \textbf{succs}\ t_2);\ !t_2\ r$ [Definition of ; ,!]

Base Case $t_1\ r = \bot$
$= \bot$ [Definition of $*,\bowtie/$]
                    [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'\ \bot$ [Definition of $*,\bowtie/$]
                    [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * \bot)$ [Definition of $*,\bowtie/$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * \bullet t_1\ r)$ [Assumption]
$= !(t_1;\ \textbf{succs}\ t_2);\ !t_2\ r$ [Definition of ; ,!]

Inductive Step on $t_1\ r$:
Suppose
$head' \bullet \bowtie/ \bullet t_2 * xs = \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * xs)$
Prove that
$head' \bullet \bowtie/ \bullet t_2 * [x] \stackrel{\bowtie}{} xs = \bowtie/ \bullet (head'\ t_2) * \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * [x] \stackrel{\bowtie}{} xs)$

$head' \bullet \bowtie/ \bullet t_2 * [x] \stackrel{\bowtie}{} xs$
$= head' \bullet \bowtie/ \bullet [t_2\ x] \stackrel{\bowtie}{} t_2 * xs$ [Definition of $*$]
$= head'(t_2\ x \stackrel{\bowtie}{} \bowtie/ t_2 * xs)$ [Definition of $\bowtie/$]

Case $t_2\ x \neq [\ ]$ and $t_2\ x \neq \bot$
$= head'(t_2\ x)$ [Property of $head'$]
$= \bowtie/ \bullet head'(t_2\ x)$ [Definition of $\bowtie/$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet [x]$ [Definition of $*$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'([x])$ [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2) * \bullet head'(\textbf{skip}\ x)$ [Definition of $\textbf{skip}$]

127

$= \bowtie/ \bullet (head'\ t_2)* \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * [x]^{\bowtie} xs)$ \hfill [Definition of $\bowtie/$]

\hfill [Definition of **succs**, $*$]

$= !(t_1;\ \textbf{succs}\ t_2);\ !t_2\ r$ \hfill [Definition of ; ,!]

Case $t_2\ x = [\ ]$
$= head' \bullet \bowtie/ \bullet t_2 * xs$ \hfill [Functional Composition]
$= \bowtie/ \bullet (head'\ t_2)* \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * xs)$ \hfill [Assumption]
$= !(t_1;\ \textbf{succs}\ t_2);\ !t_2\ r$ \hfill [Definition of ; ,!]

Case $t_2\ x = \bot$
$= \bot$ \hfill [Definition of $\bowtie$,$head'$]
$= \bowtie/ \bullet head'(\bot)$ \hfill [Definition of $\bowtie/$,$head'$]
$= \bowtie/ \bullet (head'\ t_2) * \bot$ \hfill [Definition of $*$]
$= \bowtie/ \bullet (head'\ t_2)* \bullet head'(\bot)$ \hfill [Definition of $head'$]
$= \bowtie/ \bullet (head'\ t_2)* \bullet head'(\bowtie/ \bullet (\textbf{succs}\ t_2) * [x]^{\bowtie} xs)$ \hfill [Assumption]
$= !(t_1;\ \textbf{succs}\ t_2);\ !t_2\ r$ \hfill [Definition of ; ,!]

\hfill $\square$

Law 27. $\quad \textbf{succs}(t_1 \mid t_2) =!(\textbf{succs}\ t_1 \mid \textbf{succs}\ t_2)$

$\textbf{succs}(t_1 \mid t_2)\ r$

Case $t_1\ r = [\ ]$ and $t_2\ r = [\ ]$
$= \textbf{fail}\ r$ \hfill [Definition of **succs**]
$= [\ ]$ \hfill [Definition of **fail**]
$= head'([\ ])$ \hfill [Definition of $head'$]
$= head'(\bowtie/ [\ ])$ \hfill [Definition of $\bowtie/$]
$= head'(\bowtie/ [[\ ],[\ ]])$ \hfill [Definition of $\bowtie/$]
$= head'(\bowtie/ [\textbf{fail}\ r, \textbf{fail}\ r])$ \hfill [Definition of **fail**]
$= head'(\bowtie/ [(\textbf{fail} \lhd\ t_1\ r = [\ ] \rhd \textbf{skip})\ r,$
$\qquad\qquad (\textbf{fail} \lhd\ t_2\ r = [\ ] \rhd \textbf{skip})\ r])$ \hfill [Definition of $\lhd\ \rhd$]

\hfill [Assumption]

$= head'(\bowtie/ \bullet [(\textbf{fail} \lhd\ t_{1\_} = [\ ] \rhd \textbf{skip}),$
$\qquad\qquad (\textbf{fail} \lhd\ t_{2\_} = [\ ] \rhd \textbf{skip})\ r]^{\circ}\ r$ \hfill [Definition of $\circ$]
$= !(\textbf{succs}\ t_1 \mid \textbf{succs}\ t_2)$ \hfill [Definition of **succs**]

\hfill [Definition of $\mid, head'$]

Case $t_1\ r = \bot$ or $t_2\ r = \bot$
$= \bot$ \hfill [Definition of **succs**]
$= head'(\bot)$ \hfill [Definition of $head'$]

128

$= \ !(\mathbf{succs} \ t_1 \mid \mathbf{succs} \ t_2) \ r$ [Assumption]
                                                                  [Definition of $\mid$,**succs**]

Other Cases
$= \mathbf{skip} \ r$ [Definition of **succs**]
$= [r]$ [Definition of **skip**]
$= head'([r])$ [Definition of $head'$]

Case $t_1 \ r \neq [\,] $ and $t_2 \ r = [\,]$
$= head'(\otimes/[[r], [\,]])$ [Definition of $\otimes/$]
$= head'(\otimes/[\mathbf{skip} \ r, \mathbf{fail} \ r])$ [Definition of **fail**, **skip**]

Case $t_1 \ r = [\,]$ and $t_2 \ r \neq [\,]$
$= head'(\otimes/[[\,], [r]])$ [Definition of $\otimes/$]
$= head'(\otimes/[\mathbf{fail} \ r, \mathbf{skip} \ r])$ [Definition of **fail**, **skip**]

Case $t_1 \ r \neq [\,]$ and $t_2 \ r \neq [\,]$
$= head'(\otimes/[[r], [r]])$ [Definition of $\otimes/$]
$= head'(\otimes/[\mathbf{skip} \ r, \mathbf{skip} \ r])$ [Definition of **fail**, **skip**]

In the three cases above, we have that
$= head'(\otimes/ \ [(\mathbf{fail} \lhd \ t_1 \ r = [\,] \rhd \mathbf{skip}) \ r,$
                $(\mathbf{fail} \lhd \ t_2 \ r = [\,] \rhd \mathbf{skip}) \ r])$ [Definition of $\lhd \ \rhd$]
                                                                    [Assumption]

$= head'(\otimes/ \bullet [(\mathbf{fail} \lhd \ t_{1\_} = [\,] \rhd \mathbf{skip}),$
                $(\mathbf{fail} \lhd \ t_{2\_} = [\,] \rhd \mathbf{skip}) \ r]^{\circ} \ r)$ [Definition of $\circ$]
$= \ !(\mathbf{succs} \ t_1 \mid \mathbf{succs} \ t_2)$ [Definition of **succs**, $\mid$]
                                                                [Definition of $head'$]

                                                                $\square$

Law 28. $\mathbf{fails}(t_1 \mid t_2) = \mathbf{fails} \ t_1; \ \mathbf{fails} \ t_2$

$\mathbf{fails}(t_1 \mid t_2) \ r$

Case $t_1 \ r = [\,]$ and $t_2 \ r = [\,]$
$= \mathbf{fail} \ r$ [Assumption]
$= [\,]$ [Definition of **fail**]
$= \otimes/[\,]$ [Definition of $\otimes/$]
$= \otimes/ \bullet \mathbf{fail} * [\,]$ [Definition of $*$]
$= \otimes/ \bullet \mathbf{fail}* \bullet \mathbf{fail}r$ [Definition of **fail**]

$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet (\textbf{skip} \lhd t_2{}_- = [\,] \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_1{}_- = [\,] \rhd \textbf{fail})\ r \qquad \text{[Definition of } \lhd \rhd]$$
$$= \textbf{fails}\ t_1;\ \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{fails}]$$

Case $t_1\ r = \bot$ or $t_2\ r = \bot$
$$= \bot \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } |, \textbf{fails}]$$
$$= \textbf{fails}\ t_1;\ \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{fails},;\,]$$

Case $t_1\ r \neq [\,]$ and $t_2\ r = [\,]$
$$= \textbf{skip}\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Assumption]}$$
$$= [r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{skip}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/[r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } {}^{\infty\!\!\!\!\infty}\!/]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{skip} * [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } *]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{skip} * \bullet \textbf{fail}\, r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } \textbf{fail}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet (\textbf{skip} \lhd t_2{}_- = [\,] \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_1{}_- = [\,] \rhd \textbf{fail})\ r \qquad \text{[Definition of } \lhd \rhd]$$
$$= \textbf{fails}\ t_1;\ \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{fails}]$$

Case $t_1\ r = [\,]$ and $t_2\ r \neq [\,]$
$$= \textbf{fail}\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{[Assumption]}$$
$$= [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{[Definition of } \textbf{fail}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/[\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } {}^{\infty\!\!\!\!\infty}\!/]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{fail} * [r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } *]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{fail} * \bullet \textbf{skip}\, r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } \textbf{skip}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet (\textbf{skip} \lhd t_2{}_- = [\,] \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_1{}_- = [\,] \rhd \textbf{fail})\ r \qquad \text{[Definition of } \lhd \rhd]$$
$$= \textbf{fails}\ t_1;\ \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{fails}]$$

Case $t_1\ r \neq [\,]$ and $t_2\ r \neq [\,]$
$$= \textbf{skip}\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Assumption]}$$
$$= [r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{skip}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/[r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } {}^{\infty\!\!\!\!\infty}\!/]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{skip} * [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } *]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet \textbf{skip} * \bullet \textbf{skip}\, r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } \textbf{skip}]$$
$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet (\textbf{skip} \lhd t_2{}_- = [\,] \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_1{}_- = [\,] \rhd \textbf{fail})\ r \qquad \text{[Definition of } \lhd \rhd]$$
$$= \textbf{fails}\ t_1;\ \textbf{fails}\ t_2\ r \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \textbf{fails}]$$

$\square$

Law 29. $\textbf{succs}\ s;\ \textbf{succs}(s;\ t) = \textbf{succs}(s;\ t)$

$\textbf{succs}\ s;\ \textbf{succs}\ (s;\ t)\ r$

$$= {}^{\infty\!\!\!\!\infty}\!/ \bullet (\textbf{succs}\ (s;\ t)) * \bullet \textbf{succs}\ s \qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } *]$$

Case $s\ r = [\,]$

$= {}^{\bowtie}\!/\ \bullet\ (\mathbf{succs}\ (s;\ t)) * [\,]$	[Definition of **succs**]
	[Definition of **fail**]
$= [\,]$	[Definition of $*,\ {}^{\bowtie}\!/]$
$= \mathbf{fail}\ r$	[Definition of **fail**]
$= (\mathbf{fail} \triangleleft [\,] = [\,] \triangleright \mathbf{skip})\ r$	[Definition of $\triangleleft\ \triangleright$]
$= (\mathbf{fail} \triangleleft t * [\,] = [\,] \triangleright \mathbf{skip})\ r$	[Definition of $*$]
$= (\mathbf{fail} \triangleleft t* \bullet\ s\ r = [\,] \triangleright \mathbf{skip})\ r$	[Assumption]
$= \mathbf{succs}\ (s;\ t)\ r$	[Definition of $;$]
	[Definition of **succs**]

Case $s\ r \neq [\,]$

$= {}^{\bowtie}\!/\ \bullet\ (\mathbf{succs}\ (s;\ t)) * [r]$	[Definition of **succs**]
	[Definition of **skip**]
$= \mathbf{succs}\ (s;\ t)\ [r]$	[Definition of ${}^{\bowtie}\!/]$
	[Definition of $*$]

Case $s\ r = \bot$

$= \bot$	[Definition of **succs**]
	[Definition of $;$]
$= \mathbf{succs}\ (s;\ t)r$	[Definition of $;$]
	[Definition of **succs**]

$\square$

Law 30. $\mathbf{fails}\ s = \mathbf{fails}\ s;\ \mathbf{fails}(s;\ t)$

$\mathbf{fails}\ s\ r$

$= (\mathbf{skip} \triangleleft s\ r = [\,] \triangleright \mathbf{fail})\ r$	[Definition of **fail**]

Case $s\ r = [\,]$

$= \mathbf{skip}\ r$	[Definition of $\triangleleft\ \triangleright$]
$= [r]$	[Definition of **skip**]
$= {}^{\bowtie}\!/\ [r]$	[Definition of ${}^{\bowtie}\!/]$
$= {}^{\bowtie}\!/\ \bullet\ \mathbf{skip}\ r$	[Definition of **skip**]
$= {}^{\bowtie}\!/\ \bullet\ \mathbf{skip} * [r]$	[Definition of $*$]
$= {}^{\bowtie}\!/\ \bullet\ \mathbf{skip}* \bullet\ \mathbf{skip}\ r$	[Definition of **skip**]
$= {}^{\bowtie}\!/\ \bullet\ (\mathbf{skip} \triangleleft [\,] = [\,] \triangleright \mathbf{fail})* \bullet\ (\mathbf{skip} \triangleleft\ <> = <>\ \triangleright \mathbf{fail})\ r$	[Definition of $\triangleleft\ \triangleright$]

131

$$= \,^{\otimes}\!/ \bullet (\mathbf{skip} \lhd \,^{\otimes}\!/ \bullet t \ast \bullet \, s \, r = [\,] \rhd \mathbf{fail}) \ast \bullet$$
$$(\mathbf{skip} \lhd s \, r = [\,] \rhd \mathbf{fail}) \; r \qquad\qquad\qquad \text{[Definition of } ^{\otimes}\!/, \ast]$$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Assumption]

$= \mathbf{fails} \; s; \; \mathbf{fails}(s; \; t) \qquad\qquad\qquad\qquad\qquad$ [Definition of $\mathbf{fails}$, ;]

Case $s \, r \neq [\,]$

$= \mathbf{fail} \; r \qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $\lhd \rhd$]

$= [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $<>$]

$= \,^{\otimes}\!/ \; [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $^{\otimes}\!/$]

Let $f = \mathbf{skip}$ or $f = \mathbf{fail}$

$= \,^{\otimes}\!/ \bullet f \ast [\,] > \qquad\qquad\qquad\qquad\quad$ [Definition of $\mathbf{skip}$]

$= \,^{\otimes}\!/ \bullet f \ast \bullet \mathbf{fail} \; r \qquad\qquad\qquad\qquad$ [Definition of $\mathbf{fail}$]

$= \,^{\otimes}\!/ \bullet (\mathbf{skip} \lhd \,^{\otimes}\!/ \bullet t \ast \bullet \, s \, r = [\,] \rhd \mathbf{fail}) \ast \bullet$
$\qquad (\mathbf{skip} \lhd \,^{\otimes}\!/ \bullet \, s \, r = [\,] \rhd \mathbf{fail}) \; r \qquad$ [Definition of $\lhd \rhd$]

$= \,^{\otimes}\!/(\mathbf{fails} \; (s; \; t)) \ast \bullet \mathbf{fails} \; s \qquad\qquad$ [Definition of ; , $\mathbf{fail}$]

$= \mathbf{fails} \; s; \; \mathbf{fails}(s; \; t) \qquad\qquad\qquad\qquad$ [Definition of ;]

Case $s \, r = \perp$

$= \perp \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $\mathbf{succs}$, ;]

$= \mathbf{fails} \; s; \; \mathbf{fails}(s; \; t) \; r \qquad\qquad\qquad\;$ [Definition of ; ,$\mathbf{fails}$]

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Law 31. $!s; \; \mathbf{fails} \; t = \mathbf{fails}(!s; \; t); \; !s$

$!s; \; \mathbf{fails} \; t \; r$

$= !s; \; \mathbf{fails} \; t \; r$

$= \,^{\otimes}\!/ \bullet (\mathbf{fails} \; t) \ast \bullet \, head'(s \, r) \qquad\qquad$ [Definition of $!, head'$]

Base Case $s \, r = [\,]$

$= \,^{\otimes}\!/ \bullet (\mathbf{fails} \; t) \ast \bullet \, head'([\,]) \qquad\qquad$ [Assumption]

$= \,^{\otimes}\!/ \bullet (\mathbf{fails} \; t) \ast [\,] \qquad\qquad\qquad\quad$ [Definition of $head'$]

$= \,^{\otimes}\!/[\,] \qquad\qquad\qquad\qquad\qquad\qquad\quad$ [Definition of $\ast$]

$= \,^{\otimes}\!/ \bullet head' \; [\,] \qquad\qquad\qquad\qquad\quad$ [Definition of $head'$]

$= \,^{\otimes}\!/ \bullet head'(s \, r) \qquad\qquad\qquad\qquad$ [Assumption]

$= \,^{\otimes}\!/ \bullet (head' \; s) \ast [r] \qquad\qquad\qquad$ [Definition of $\ast$]

$= \,^{\otimes}\!/ \bullet (head' \; s) \ast \bullet \mathbf{skip} \; r \qquad\qquad$ [Definition of $\mathbf{skip}$]

$= \,^{\otimes}\!/ \bullet (head' \; s) \ast \bullet (\mathbf{skip} \lhd [\,] = [\,] \rhd \mathbf{fail}) \; r \qquad$ [Definition of $\lhd \rhd$]

$= \,^{\otimes}\!/ \bullet (head' \; s) \ast \bullet (\mathbf{skip} \lhd \,^{\otimes}\!/ \bullet t \ast \bullet \, head'(s \, r) = [\,] \rhd \mathbf{fail}) \; r$ [Definition of $^{\otimes}\!/, head'$]

$= \mathbf{fails}(!s; \; t); \; !s \; r \qquad\qquad\qquad\qquad$ [Assumption]

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ [Definition of ; , !]

132

Base Case $s\ r = \bot$
$= \bowtie\!/ \bullet (\textbf{fails}\ t) * \bullet\ head'(\bot)$ [Assumption]
$= \bowtie\!/ \bot$ [Definition of $*,head'$]
$= \bot$ [Definition of $\bowtie\!/$]
$= \textbf{fails}(!s;\ t);\ !s\ r$ [Definition of $\textbf{fails}$]

Inductive Step on $s\ r$:
Suppose
$\bowtie\!/ \bullet (\textbf{fails}\ t) * \bullet\ head'(xs) = \bowtie\!/ \bullet (head'\ s) * \bullet (\textbf{skip} \lhd \bowtie\!/ \bullet\ t * \bullet\ head'(xs) = [\ ] \rhd \textbf{fail})\ r$
Prove that
$\bowtie\!/ \bullet (\textbf{fails}\ t) * \bullet\ head'([x] \bowtie xs)$
$\quad = \bowtie\!/ \bullet (head'\ s) * \bullet (\textbf{skip} \lhd \bowtie\!/ \bullet\ t * \bullet\ head'([x] \bowtie xs) = [\ ] \rhd \textbf{fail})\ r$

$\bowtie\!/ \bullet (\textbf{fails}\ t) * \bullet\ head'([x] \bowtie xs)$
$= \bowtie\!/ \bullet (\textbf{fails}\ t) * [x]$ [Definition of $head'$]
$= \bowtie\!/(\textbf{fails}\ t\ x)$ [Definition of $*$]

Case $t\ x = [\ ]$
$= \bowtie\!/(\textbf{fails}\ t\ x)$ [Definition of $*$]
$= [x]$ [Definition of $\textbf{fails},\bowtie\!/$]
$= \bowtie\!/ head'([x] \bowtie xs)$ [Definition of $\bowtie\!/,head'$]
$= \bowtie\!/ head'(s\ r)$ [Assumption]
$= \bowtie\!/(head'\ s) * [r]$ [Definition of $*$]
$= \bowtie\!/ \bullet (head'\ s) * \bullet\ \textbf{skip}\ [r]$ [Definition of $\textbf{skip}$]
$= \bowtie\!/ \bullet (head'\ s) * \bullet (\textbf{skip} \lhd \bowtie\!/ \bullet [\ ] = [\ ] \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= \bowtie\!/ \bullet (head'\ s) * \bullet (\textbf{skip} \lhd \bowtie\!/ \bullet\ t * \bullet\ head'([x] \bowtie xs) = [\ ] \rhd \textbf{fail})\ r$
  [Definition of $head',*$]
$= \bowtie\!/ \bullet (head'\ s) * \bullet (\textbf{skip} \lhd \bowtie\!/ \bullet\ t * \bullet\ head'([x] \bowtie xs) = [\ ] \rhd \textbf{fail})\ r$
  [Definition of $\bowtie\!/,head'$]

Case $t\ x = \bot$
$= \bot$ [Definition of $\textbf{fails},\bowtie\!/$]
$= \bowtie\!/ head'(\bot)$ [Definition of $\bowtie\!/,head'$]
$= \bowtie\!/(head'\ s) * \bot$ [Definition of $*$]
$= \textbf{fails}(!s;\ t);\ !s\ r$ [Assumption]
  ,Definition ; ,!]

Case $t\ x = ys$
$= \bowtie\!/(\textbf{fails}\ t\ x)$ [Definition of $*$]
$= [\ ]$ [Definition of $\textbf{fails},\bowtie\!/$]
$= \bowtie\!/ head'([\ ])$ [Definition of $\bowtie\!/,head'$]
$= \bowtie\!/ head'(s\ r)$ [Assumption]

$= {}^{\bowtie}\!/(head'\ s) * [\,]$ [Definition of $*$]

$= {}^{\bowtie}\!/\bullet (head'\ s) * \bullet \mathbf{fail}\ [r]$ [Definition of $\mathbf{skip}$]

$= {}^{\bowtie}\!/\bullet (head'\ s) * \bullet (\mathbf{skip} \lhd {}^{\bowtie}\!/ \bullet ys = [\,] \rhd \mathbf{fail})\ r$

     [Definition of $\lhd\ \rhd$]

$= {}^{\bowtie}\!/\bullet (head'\ s) * \bullet (\mathbf{skip} \lhd {}^{\bowtie}\!/ \bullet t * \bullet head'([x] \mathbin{\bowtie} xs) = [\,] \rhd \mathbf{fail})\ r$

     [Assumption]

     [Definition of $head', *$]

                 $\square$

Law 32. $\mathbf{fails}(\mathbf{fails}(s;\ t)) = \mathbf{succs}\ s \mid (\mathbf{fails}\ s;\ \mathbf{fails}\ t)$

$\mathbf{fails}(\mathbf{fails}(s;\ t))r$

$= (\mathbf{abort} \lhd \mathbf{fails}(\mathbf{fails}(s;\ t))\ r = \bot \rhd (\mathbf{skip} \lhd \mathbf{fails}\ s;\ t\ r = [\,] \rhd \mathbf{fail}))\ r$

     [Definition of $\mathbf{fails}$]

$= (\mathbf{abort} \lhd \mathbf{fails}(\mathbf{fails}(s;\ t))\ r = \bot \rhd (\mathbf{skip} \lhd (t * \bullet (\mathbf{skip} \lhd s\ r = [\,] \rhd \mathbf{fail})\ r = [\,] \rhd \mathbf{fail})))\ r$

     [Definition of $\mathbf{fails}$]

Case $s\ r = \bot$

$= \bot$ [Definition of $\lhd\ \rhd$]

$= \mathbf{succs}\ s \mid (\mathbf{fails}\ s;\ \mathbf{fails}\ t)\ r$ [Definition of $\mathbf{fails}$]

     [Definition of $\mathbf{succs}, \mid$]

Case $s\ r = [\,]$

$= (\mathbf{skip} \lhd t * [r] = [\,] \rhd \mathbf{fail})\ r$ [Definition of $\lhd\ \rhd$]

$= (\mathbf{skip} \lhd t\ r = [\,] \rhd \mathbf{fail})\ r$ [Definition of $*$]

$= \mathbf{fails}\ t\ r$ [Definition of $\mathbf{fails}$]

$= {}^{\bowtie}\!/[[\,], \mathbf{fails}\ t\ r]$ [Definition of ${}^{\bowtie}\!/$]

$= {}^{\bowtie}\!/[\mathbf{succs}\ s\ r, \mathbf{fails}\ t\ r]$ [Assumption]

$= {}^{\bowtie}\!/[\mathbf{succs}\ s\ r, {}^{\bowtie}\!/\bullet (\mathbf{fails}\ t) * \bullet \mathbf{fails}\ s\ r]$ [Definition of $*$]

     [Assumption]

$= {}^{\bowtie}\!/[\mathbf{succs}\ s, {}^{\bowtie}\!/\bullet (\mathbf{fails}\ t) * \bullet \mathbf{fails}\ s]^{\circ}$ [Definition of $\circ$]

$= \mathbf{succs}\ s \mid (\mathbf{fails}\ s;\ \mathbf{fails}\ t)\ r$ [Definition of $\mid$]

Case $s\ r \neq [\,]$

$= (\mathbf{skip} \lhd t * \bullet \mathbf{fails}\ r = [\,] \rhd \mathbf{fail})\ r$ [Definition of $\lhd\ \rhd$]

$= \mathbf{skip}\ r$ [Definition of $\mathbf{fail}, *, \lhd\ \rhd$]

$= [r]$ [Definition of $\mathbf{skip}$]

$= {}^{\bowtie}\!/[[r], [\,]]$ [Definition of ${}^{\bowtie}\!/$]

$= {}^{\bowtie}\!/[\mathbf{skip}\ r, {}^{\bowtie}\!/\bullet (\mathbf{fails}\ t) * [\,]]$ [Definition of $\mathbf{skip}, *$]

$= \text{⨳}/[\textbf{fail} \lhd s \ r = [\,] \rhd \textbf{skip}) \ r, \text{⨳}/ \bullet (\textbf{fails} \ t) * \bullet$
$\quad\quad (\textbf{skip} \lhd s \ r = [\,] \rhd \textbf{fail}) \ r]$ $\quad\quad\quad\quad$ [Definition of $\lhd \ \rhd$]
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Assumption]
$= \text{⨳}/[\textbf{succs} \ s \ r, \text{⨳}/ \bullet (\textbf{fails} \ t) * \bullet \ \textbf{fails} \ s \ r]$ $\quad$ [Definition of $\textbf{fails}$]
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\textbf{succs}$]
$= \text{⨳}/[\textbf{succs} \ s \ r, \textbf{fails} \ s; \ \textbf{fails} \ t \ r]$ $\quad\quad\quad$ [Definition of ;]
$= \text{⨳}/[\textbf{succs} \ s, \textbf{fails} \ s; \ \textbf{fails} \ t]^\circ$ $\quad\quad\quad$ [Definition of $^\circ$]
$= \textbf{succs} \ s \mid (\textbf{fails} \ s; \ \textbf{fails} \ t) \ r$ $\quad\quad\quad$ [Definition of $||$]

$\square$

Law 33. $\textbf{fails}(s; \ \textbf{fails} \ t) = \textbf{fails} \ s \mid \textbf{succs}(s; \ t)$

$\textbf{fails}(s; \ \textbf{fails} \ t) \ r$

$= (\textbf{abort} \lhd \textbf{fails}(s; \ \textbf{fails} \ t) \ r = \bot \rhd (\textbf{skip} \lhd s; \ \textbf{fails} \ t \ r = [\,] \rhd \textbf{fail})) \ r$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\textbf{fails}$]
$= (\textbf{abort} \lhd \textbf{fails}(s; \ \textbf{fails} \ t) \ r = \bot \rhd (\textbf{skip} \lhd \text{⨳}/ \bullet (\textbf{fails} \ t) * \bullet \ s \ r = [\,] \rhd \textbf{fail})) \ r$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of ;]

Case $s \ r = \bot$
$= \bot$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\lhd \ \rhd$]
$= \textbf{fails} \ s \mid \textbf{succs}(s; \ t) \ r$ $\quad\quad\quad$ [Definition of $\textbf{fails}, |, ; $]
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\textbf{succs}$]

Case $s \ r = [\,]$
$= (\textbf{skip} \lhd [\,] = [\,] \rhd \textbf{fail}) \ r$ $\quad\quad\quad$ [Definition of $*, \text{⨳}/$]
$= \textbf{skip} \ r$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\lhd \ \rhd$]
$= [r]$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\textbf{skip}$]
$= \text{⨳}/[[r], [\,]]$ $\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\text{⨳}/$]
$= \text{⨳}/[\textbf{fails} \ s \ r, \textbf{succs} \ s \ r]$ $\quad\quad\quad$ [Definition of $\textbf{fails}$]
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ [Definition of $\textbf{succs}$]
$= \text{⨳}/[\textbf{fails} \ s \ r, (\textbf{fail} \lhd s \ r = [\,] \rhd \textbf{skip}) \ r]$ $\quad$ [Definition of $\textbf{succs}$]
$= \text{⨳}/[\textbf{fails} \ s \ r, (\textbf{fail} \lhd \text{⨳}/ \bullet t * \bullet \ s \ r = [\,] \rhd \textbf{skip}) \ r]$ $\quad$ [Definition of $*, \text{⨳}/$]
$= \text{⨳}/[\textbf{fails} \ s \ r, \textbf{succs} (\text{⨳}/ \bullet t * \bullet \ s) \ r]$ $\quad$ [Definition of $\textbf{succs}$]
$= \text{⨳}/[\textbf{fails} \ s \ r, \textbf{succs} (s; \ t) \ r]$ $\quad\quad$ [Definition of ;]
$= \text{⨳}/[\textbf{fails} \ s, \textbf{succs} (s; \ t) >^\circ$ $\quad\quad\quad$ [Definition of $^\circ$]
$= \textbf{fails} \ s \mid \textbf{succs}(s; \ t) \ r$ $\quad\quad\quad$ [Definition of $||$]

Case $s \ r \neq [\,]$
$= (\textbf{skip} \lhd \text{⨳}/ \bullet (\textbf{fails} \ t) * \bullet \ s \ r = [\,] \rhd \textbf{fail}) \ r$

135

We have that
$$= (\textbf{fail} \lhd \bowtie\!/ \bullet t* \bullet s \ r = [\,] \rhd \textbf{skip})\ r$$
$$= \bowtie \bullet [[\,], (\textbf{fail} \lhd \bowtie\!/ \bullet t* \bullet s \ r = [\,] \rhd \textbf{skip})\ r] \qquad \text{[Definition of } \bowtie\!/]$$
$$= \bowtie \bullet [\textbf{fails}\ s\ r, \textbf{succs}\ (s;\ t)\ r] \qquad \text{[Definition of ;]}$$
$$\qquad \text{[Definition of \textbf{succs}]}$$
$$= \bowtie\!/[\textbf{fails}\ s, \textbf{succs}\ (s;\ t)]^{\circ} \qquad \text{[Definition of }^{\circ}]$$
$$= \textbf{fails}\ s \mid \textbf{succs}(s;\ t)\ r \qquad \text{[Definition of }|]$$

$$\square$$

Law 34^*. $\quad \textbf{fails}\ s;\ \textbf{succs}(s;\ t) = \textbf{fail}$

$\textbf{fails}\ s;\ \textbf{succs}(s;\ t)\ r$

$$= \bowtie\!/ \bullet (\textbf{succs}(s;\ t))* \bullet \textbf{fails}\ s\ r$$

Case $s\ r = [\,]$
$$= \bowtie\!/ \bullet \textbf{succs}\ (s;\ t)\ r \qquad \text{[Definition of \textbf{fail}, \textbf{skip}, *]}$$
$$= \bowtie\!/ \bullet \textbf{succs}\ (\bowtie\!/ \bullet t* \bullet s)\ r \qquad \text{[Definition of ;]}$$
$$= [\,] \qquad \text{[Definition of \textbf{succs}]}$$
$$\qquad \text{[Assumption]}$$
$$= \textbf{fail}\ r \qquad \text{[Definition of \textbf{fail}]}$$

Case $s\ r \neq [\,]$
$$= \bowtie\!/ \bullet (\textbf{succs}\ (s;\ t)) * [\,] \qquad \text{[Definition of \textbf{fails}, \textbf{fail}]}$$
$$= [\,] \qquad \text{[Definition of }*]$$
$$= \textbf{fail}\ r \qquad \text{[Definition of \textbf{fail}]}$$

$$\square$$

Law 35. $\quad \textbf{fails}(t;\ d) = \textbf{fails}(t;\ \textbf{succs}\ d)$

$\textbf{fails}\ (t;\ d)\ r$

Case $s\ r = \bot$
$$= \bot \qquad \text{[Definition of \textbf{fails}]}$$
$$= \textbf{fails}\ (t;\ \textbf{succs}\ d)\ r \qquad \text{[Definition of \textbf{fails}]}$$

$$= (\textbf{skip} \lhd \bowtie\!/ \bullet d* \bullet t\ r = [\,] \rhd \textbf{fail})\ r \qquad \text{[Definition of \textbf{fails}]}$$

136

Case $s\ r = [\,]$
$= (\mathbf{skip} \lhd [\,] = [\,] \rhd \mathbf{fail})\ r$ [Definition of $\otimes$, $*$]
$= (\mathbf{skip} \lhd \otimes\!/ \bullet (\mathbf{succs}\ d)* \bullet t\ r = [\,] \rhd \mathbf{fail})\ r$ [Definition of $*$]
                                                     [Assumption]
$= \mathbf{fails}\ (t;\ \mathbf{succs}\ d)\ r$ [Definition of $\mathbf{fails}$]

Case $s\ r \neq [\,]$
$= (\mathbf{skip} \lhd \otimes\!/ \bullet (\mathbf{succs}\ d)* \bullet t\ r = [\,] \rhd \mathbf{fail})\ r$ [Lemma D.6.16]
$= \mathbf{fails}\ (t;\ \mathbf{succs}\ d)\ r$ [Definition of $\mathbf{fails}$]

                                                                     $\square$

Law 36. $!s;\ \mathbf{succs}\ t = \mathbf{succs}\ (!s;\ t);\ !s$

$!s;\ \mathbf{succs}\ t\ r$

$= \otimes\!/ \bullet (\mathbf{succs}\ t)* \bullet head'(s\ r)$ [Definition of ;]

Base Case $s\ r = [\,]$
$= [\,]$ [Definition of $head'$]
                                                     [Definition of $*$, $\otimes\!/$]
$= \otimes\!/ \bullet (head'\ s) * [\,]$ [Definition of $head'$]
                                                       [Definition of $*$, $\otimes\!/$]
$= \otimes\!/ \bullet (head'\ s)* \bullet \mathbf{fail}\ r[\,]$ [Definition of $\mathbf{fail}$]
$= \otimes\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd [\,] = [\,] \rhd \mathbf{skip})\ r$ [Definition of $\lhd\ \rhd$]
$= \otimes\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \otimes\!/ \bullet t* \bullet head'(s\ r) = [\,] \rhd \mathbf{skip})\ r$
                                                                    [Definition of $head'$]
                                                                               [Definition of $*$, $\otimes\!/$]
$= \mathbf{succs}(!s;\ t);\ !s$ [Definition of ; , !, $\mathbf{succs}$]

Base Case $s\ r = \bot$
$= \bot$ [Definition of $head'$]
                                                     [Definition of $*$, $\otimes\!/$]
$= \otimes\!/ \bullet (head'\ s) * \bot$ [Definition of $head'$]
                                                       [Definition of $*$, $\otimes\!/$]
$= \mathbf{succs}(!s;\ t);\ !s$ [Definition of ; , !, $\mathbf{succs}$]

Inductive step on $s\ r$:
Suppose
$\otimes\!/ \bullet (\mathbf{succs}\ t)* \bullet head'(xs) = \otimes\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \otimes\!/ \bullet t* \bullet head'(xs) = [\,] \rhd \mathbf{skip})\ r$
Prove that

137

$$⋈\!/ \bullet (\mathbf{succs}\ t)* \bullet head'([x] \overset{⋈}{} xs)$$
$$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t* \bullet head'([x] \overset{⋈}{} xs) = [\,] \vartriangleright \mathbf{skip})\ r$$

$$⋈\!/ \bullet (\mathbf{succs}\ t)* \bullet head'([x] \overset{⋈}{} xs)$$

$= ⋈\!/ \bullet (\mathbf{succs}\ t) * [x]$	[Definition of $head'$]
$= ⋈\!/ \bullet \mathbf{succs}\ t\ x$	[Definition of $*$]

Case $t\ x = \bot$

$= \bot$	[Definition of $\mathbf{succs}$]
	[Definition of $\mathbf{skip}$, $⋈\!/$]
$= ⋈\!/ \bullet (head'\ s) * \bot$	[Definition of $*$]

$$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{abort} \vartriangleleft ⋈\!/ \bullet t* \bullet head'([x] \overset{⋈}{} xs)\ r = \bot \vartriangleright$$
$$(\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t* \bullet head'([x] \overset{⋈}{} xs) = [\,] \vartriangleright \mathbf{skip}))\ r$$

	[Definition of $head'$,$*$]
$= \mathbf{succs}(!s;\ t);\ !s$	[Definition of $;$, $!, \mathbf{succs}$]

Case $t\ x = [\,]$

$= [\,]$	[Definition of $\mathbf{succs}$]
	[Definition of $\mathbf{skip}$, $⋈\!/$]
$= ⋈\!/ \bullet (head'\ s) * [\,]$	[Definition of $*$]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft [\,] = [\,] \vartriangleright \mathbf{skip})\ r$	[Definition of $\vartriangleleft \vartriangleright$]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t\ x = [\,] \vartriangleright \mathbf{skip})\ r$	[Assumption]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t* \bullet head'([x] \overset{⋈}{} xs) = [\,] \vartriangleright \mathbf{skip})\ r$	
	[Definition of $head'$,$*$]

Case $t\ x = ys$

$= [x]$	[Definition of $\mathbf{succs}$]
	[Definition of $\mathbf{skip}$, $⋈\!/$]
$= ⋈\!/ \bullet (head'\ s) * [r]$	[Definition of $*$,$head'$]
$= ⋈\!/ \bullet (head'\ s)* \bullet \mathbf{skip}\ r$	[Definition of $\mathbf{skip}$]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ys = [\,] \vartriangleright \mathbf{skip})\ r$	[Definition of $\vartriangleleft \vartriangleright$]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t\ x = [\,] \vartriangleright \mathbf{skip})\ r$	[Assumption]
$= ⋈\!/ \bullet (head'\ s)* \bullet (\mathbf{fail} \vartriangleleft ⋈\!/ \bullet t* \bullet head'([x] \overset{⋈}{} xs) = [\,] \vartriangleright \mathbf{skip})\ r$	
	[Definition of $head'$,$*$]

□

Law 37. **fails**(**fails** t) = **succs** t

fails(**fails** t) r

138

Case $t\ r = \perp$
$= \perp$ [Definition of **fails**]
$= \textbf{succs}\ t\ r$ [Definition of **succs**]

Otherwise
$= (\textbf{skip} \lhd \textbf{fails}\ t\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of **fails**]
$= (\textbf{skip} \lhd (\textbf{skip}\ t\ r = [\,] \rhd \textbf{fail})\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of **fails**]

Case $t\ r = [\,]$
$= (\textbf{skip} \lhd [r]\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of $\lhd\,\rhd$, **skip**]
$= \textbf{fail}\ r$ [Definition of $\lhd\,\rhd$]
$= (\textbf{fail} \lhd t\ r = [\,] > \rhd \textbf{skip})\ r$ [Definition of $\lhd\,\rhd$]
       [Assumption]
$= \textbf{succs}\ t\ r$ [Definition of **succs**]

Case $t\ r \neq [\,]$
$= (\textbf{skip} \lhd [\,]\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of $\lhd\,\rhd$, **fail**]
$= \textbf{skip}\ r$ [Definition of $\lhd\,\rhd$]
$= (\textbf{fail} \lhd t\ r = [\,] > \rhd \textbf{skip})\ r$ [Definition of $\lhd\,\rhd$]
       [Assumption]
$= \textbf{succs}\ t\ r$ [Definition of **succs**]

$\square$

Law 38^{*}.   **fails** t_1; **succs** $t_2 = $ **succs** t_2; **fails** t_1

fails t_1; **succs** $t_2\ r$

$= {}^{\infty}\!/ \bullet (\textbf{succs}\ t_2)* \bullet \textbf{fails}\ t_1\ r$
Case $t_1\ r = [\,]$ and $t_2\ r = [\,]$
$= {}^{\infty}\!/ \bullet (\textbf{succs}\ t_2) * [r]$ [Definition of **fails**]
       [Definition of **skip**]
$= {}^{\infty}\!/ \bullet [\textbf{succs}\ t_2\ r]$ [Definition of $*$]
$= {}^{\infty}\!/[[\,]]$ [Definition of **succs**]
       [Definition of **fails**]
$= [\,]$ [Definition of ${}^{\infty}\!/]$
$= {}^{\infty}\!/[\,]$ [Definition of ${}^{\infty}\!/]$
$= {}^{\infty}\!/ \bullet (\textbf{fails}\ t_1) * [\,]$ [Definition of $*$]
$= {}^{\infty}\!/ \bullet (\textbf{fails}\ t_1)* \bullet \textbf{succs}\ t_2\ r$ [Definition of **fails**]
       [Definition of **succs**]
$= \textbf{succs}\ t_2$; **fails** $t_1\ r$ [Definition of ;]

139

Case $t_1\ r = [\ ]$ and $t_2\ r \neq [\ ]$

$= \bowtie\!/ \bullet (\textbf{succs}\ t_2) * [r]$ [Definition of **fails**]

[Definition of **skip**]

$= \bowtie\!/ \bullet [\textbf{succs}\ t_2\ r]$ [Definition of $*$]

$= \bowtie\!/[[r]]$ [Definition of **succs**]

[Definition of **skip**]

$= \bowtie\!/[\textbf{fails}\ t_1\ r]$ [Definition of **fails**]

[Definition of **skip**]

$= \bowtie\!/ \bullet (\textbf{fails}\ t_1) * [r]$ [Definition of $*$]

$= \bowtie\!/ \bullet (\textbf{fails}\ t_1)* \bullet \textbf{succs}\ t_2\ r$ [Definition of **succs**]

[Definition of **skip**]

$= \textbf{succs}\ t_2;\ \textbf{fails}\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq [\ ]$ and $t_2\ r = [\ ]$

$= \bowtie\!/ \bullet (\textbf{succs}\ t_2) * [\ ]$ [Definition of **fails**]

[Definition of **fail**]

$= [\ ]$ [Definition of $*, \bowtie\!/$]

$= \bowtie\!/ \bullet (\textbf{fails}\ t_1) * [\ ]$ [Definition of $*, \bowtie\!/$]

$= \bowtie\!/ \bullet (\textbf{fails}\ t_1)* \bullet \textbf{succs}\ t_2\ r$ [Definition of **succs**]

[Definition of **fail**]

$= \textbf{succs}\ t_2;\ \textbf{fails}\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq [\ ]$ and $t_2\ r \neq [\ ]$

$= \bowtie\!/ \bullet (\textbf{succs}\ t_2) * [\ ]$ [Definition of **fails**]

[Definition of **fail**]

$= [\ ]$ [Definition of $*, \bowtie\!/$]

$= \bowtie\!/[\textbf{fails}\ t_1\ r]$ [Definition of **fails**]

[Definition of **fail**, $\bowtie\!/$]

$= \bowtie\!/ \bullet (\textbf{fails}\ t_1)* \bullet \textbf{succs}\ t_2\ r$ [Definition of **succs**]

[Definition of **skip**]

$= \textbf{succs}\ t_2;\ \textbf{fails}\ t_1\ r$ [Definition of ;]

□

Law 39^*. $\textbf{succs}\ t_1;\ \textbf{succs}\ t_2 = \textbf{succs}\ t_2;\ \textbf{succs}\ t_1$

$\textbf{succs}\ t_1;\ \textbf{succs}\ t_2\ r$

$= \bowtie\!/ \bullet (\textbf{succs}\ t_2)* \bullet \textbf{succs}\ t_1\ r$

Case $t_1\ r = [\ ]$ and $t_2\ r = [\ ]$

$$= {\Join}/ \bullet (\textbf{succs } t_2) * [\,] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{fail}]}$$
$$= [\,] \qquad\qquad \text{[Definition of } *, {\Join}/\,]$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * [\,] \qquad\qquad \text{[Definition of } *, {\Join}/\,]$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * \bullet \textbf{ succs } t_2 \ r \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{fail}]}$$
$$= \textbf{succs } t_2; \textbf{ succs } t_1 \ r \qquad\qquad \text{[Definition of ;]}$$

Case $t_1 \ r = [\,]$ and $t_2 \ r \neq [\,]$
$$= {\Join}/ \bullet (\textbf{succs } t_2) * [\,] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{fail}]}$$
$$= [\,] \qquad\qquad \text{[Definition of } *, {\Join}/\,]$$
$$= {\Join}/(\textbf{succs } t_1 \ r) \qquad\qquad \text{[Definition of } {\Join}/\,]$$
$$\text{[Definition of \textbf{succs}]}$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * [r] \qquad\qquad \text{[Definition of } *]$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * \bullet \textbf{ succs } t_2 \ r \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{skip}]}$$
$$= \textbf{succs } t_2; \textbf{ succs } t_1 \ r \qquad\qquad \text{[Definition of ;]}$$

Case $t_1 \ r \neq [\,]$ and $t_2 \ r = [\,]$
$$= {\Join}/ \bullet (\textbf{succs } t_2) * [r] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{skip}]}$$
$$= {\Join}/ [\textbf{succs } t_2 \ r] \qquad\qquad \text{[Definition of } *]$$
$$= {\Join}/ [[\,]] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{fail}]}$$
$$= [\,] \qquad\qquad \text{[Definition of } {\Join}/\,]$$
$$= {\Join}/ [\,] \qquad\qquad \text{[Definition of } {\Join}/\,]$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * [\,] \qquad\qquad \text{[Definition of } *]$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * \bullet \textbf{ succs } t_2 \ r \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{fail}]}$$
$$= \textbf{succs } t_2; \textbf{ succs } t_1 \ r \qquad\qquad \text{[Definition of ;]}$$

Case $t_1 \ r \neq [\,]$ and $t_2 \ r \neq [\,]$
$$= {\Join}/ \bullet (\textbf{succs } t_2) * [r] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{skip}]}$$
$$= {\Join}/ [\textbf{succs } t_2 \ r] \qquad\qquad \text{[Definition of } *]$$
$$= {\Join}/ [[r]] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{skip}]}$$
$$= {\Join}/ [\textbf{succs } t_1 \ r] \qquad\qquad \text{[Definition of \textbf{succs}]}$$
$$\text{[Definition of \textbf{skip}]}$$
$$= {\Join}/ \bullet (\textbf{succs } t_1) * [r] \qquad\qquad \text{[Definition of } *]$$

$= \overset{\infty}{/} \bullet (\textbf{succs } t_1)* \bullet \textbf{succs } t_2 \ r$ [Definition of **succs**]

[Definition of **skip**]

$= \textbf{succs } t_2; \textbf{ succs } t_1 \ r$ [Definition of ;]

□

Law 40. **succs**(**succs** t) = **succs** t

succs(**succs** t) r
Case $t \ r = \perp$
$= \perp$ [Definition of **succs**]
$= \textbf{succs } t \ r$ [Definition of **succs**]

Case $t \ r = [\,]$
$= (\textbf{fail} \lhd \textbf{succs } t \ r = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]
$= \textbf{fail } r$ [Definition of **succs**]

[Definition of $\lhd \rhd$]
$= \textbf{succs } t \ r$ [Definition of **succs**]

[Assumption]

Case $t_1 \ r \neq [\,]$
$= (\textbf{fail} \lhd \textbf{succs } t \ r = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]
$= \textbf{skip } r$ [Definition of **succs**]

[Definition of $\lhd \rhd$]
$= \textbf{succs } t \ r$ [Definition of **succs**]

[Assumption]

□

Law 41. **succs**(**fails** t) = **fails** t

succs(**fails** t) r
Case $t \ r = \perp$
$= \perp$ [Definition of **succs**]

[Definition of **fails**]
$= \textbf{fails } t \ r$ [Definition of **fails**]

Otherwise
$= (\textbf{fail} \lhd \textbf{fails } t \ r = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]

Case $t \ r = [\,]$

$= \mathbf{skip}\ r$ [Definition of **fails**]

[Definition of $\lhd \ \rhd$]

$= \mathbf{fails}\ t\ r$ [Definition of **fails**]

Case $t_1\ r \neq [\,]$

$= \mathbf{fail}\ r$ [Definition of **fails**]

[Definition of $\lhd \ \rhd$]

$= \mathbf{fails}\ t\ r$ [Definition of **fails**]

$\square$

Law 42. $\mathbf{succs}(t;\ d) = \mathbf{succs}(t;\ \mathbf{succs}\ d)$

$\mathbf{succs}(t;\ d)\ r$

Case $t\ r = \bot$

$= \bot$ [Definition of **succs**]

$= \mathbf{succs}(t;\ \mathbf{succs}\ d)\ r$ [Definition of ; ,**succs**]

Otherwise

$= (\mathbf{fail} \lhd \overset{\infty}{/} \bullet d* \bullet t\ r = [\,] \rhd \mathbf{skip})\ r$ [Definition of **succs**, ;]

Case $t\ r = [\,]$

$= (\mathbf{fail} \lhd \overset{\infty}{/} \bullet d* \bullet t\ r = [\,] \rhd \mathbf{skip})\ r$

We can say that this is the same as

$= (\mathbf{fail} \lhd \overset{\infty}{/} \bullet (\mathbf{succs}\ d)* \bullet t\ r = [\,] \rhd \mathbf{skip})\ r$

Since the existence of a r_n in $t\ r$ such that $d\ r_n = [\,]$ implies that, in the same way, $\mathbf{succs}\ d\ r_n = [\,]$

$= (\mathbf{fail} \lhd t;\ \mathbf{succs}\ d\ r = [\,] \rhd \mathbf{skip})\ r$ [Definition of ;]

$= \mathbf{succs}\ (t;\ \mathbf{succs}\ d)\ r$ [Definition of **succs**]

Case $t_1\ r \neq [\,]$

$= (\mathbf{fail} \lhd \overset{\infty}{/} \bullet d* \bullet t\ r = [\,] \rhd \mathbf{skip})\ r$

We can say that this is the same as

$= (\mathbf{fail} \lhd \overset{\infty}{/} \bullet (\mathbf{succs}\ d)* \bullet t\ r \neq [\,] \rhd \mathbf{skip})\ r$

Since the existence of a r_n in $t\ r$ such that $d\ r_n \neq [\,]$ implies that, in the same way, $\mathbf{succs}\ d\ r_n \neq [\,]$

$= (\mathbf{fail} \lhd t;\ \mathbf{succs}\ d\ r = [\,] \rhd \mathbf{skip})\ r$ [Definition of ;]

$= \mathbf{succs}\ (t;\ \mathbf{succs}\ d)\ r$ [Definition of **succs**]

$\square$

143

Law 43. **succs** $t =$ **succs** $!t =!$ **succs** t

First, let us prove that **succs** $t =$ **succs** $!t$. We know that $t\ r = [\,] \Leftrightarrow head'(t\ r) = [\,]$, this is $!t = [\,]$. In the same way $t\ r \neq [\,] \Leftrightarrow head'(t\ r) \neq [\,]$, this os $!t \neq [\,]$. Also, if $t\ r = \bot$ we have that $!t\ r = \bot$. This finishes this part of the proof.

Now let us prove that **succs** $!t =!$ **succs** t

succs $!t\ r$
Case $t\ r = [\,]$
$= [\,]$ [Definition of $!$, **succs**]
                    [Definition of **fail**]
$= head'\ [\,]$ [Definition of $head'$]
$= head'(\textbf{succs}\ t\ r)$ [Definition of **succs**]
                    [Definition of **fail**]
$=\ !\,\textbf{succs}\ t\ r$ [Definition of $!$]

Case $t\ r = \bot$
$= \bot$ [Definition of $!$, **succs**]
                    [Definition of **fail**]
$= head'\ \bot$ [Definition of $head'$]
$= head'(\textbf{succs}\ t\ r)$ [Definition of **succs**]
$=\ !\,\textbf{succs}\ t\ r$ [Definition of $!$]

Case $t\ r \neq [\,]$
$= <r>$ [Definition of $!$, **succs**]
                    [Definition of **skip**]
$= head'\ [r]$ [Definition of $head'$]
$= head'(\textbf{succs}\ t\ r)$ [Definition of **succs**]
                    [Definition of **skip**]
$=\ !\,\textbf{succs}\ t\ r$ [Definition of $!$]

$\square$

Law 44. **succs skip** $=$ **skip**

succs skip r
$= (\textbf{fail} \lhd \textbf{skip}\ r = [\,] \rhd \textbf{skip})\ r$ [Definition of **succs**]
$= (\textbf{fail} \lhd [r] = [\,] \rhd \textbf{skip})\ r$ [Definition of **skip**]
$= \textbf{skip}$ [Definition of $\lhd\ \rhd$]

$\square$

Law 45. **succs fail = fail**

succs fail r
$=$ (**fail** $\lhd$ **fail** $r = [\,] \rhd$ **skip**) r [Definition of **succs**]
$=$ (**fail** $\lhd [\,] = [\,] \rhd$ **skip**) r [Definition of **fail**]
$=$ **fail**   [Definition of $\lhd \rhd$]

                          $\square$

Law 46. **fails skip = fail**

fails skip r
$=$ (**skip** $\lhd$ **skip** $r = [\,] \rhd$ **fail**) r [Definition of **fails**]
$=$ (**skip** $\lhd [r] = [\,] \rhd$ **fail**) r [Definition of **skip**]
$=$ **fail**   [Definition of $\lhd \rhd$]

                          $\square$

Law 47. **fails fail = skip**

fails fail r
$=$ (**skip** $\lhd$ **fail** $r = [\,] \rhd$ **fail**) r [Definition of **fails**]
$=$ (**skip** $\lhd [\,] = [\,] \rhd$ **fail**) r [Definition of **fail**]
$=$ **skip**   [Definition of $\lhd \rhd$]

                          $\square$

Law 48. **fails** t; **fails** $t =$ **fails** t

fails t; **fails** t r
$= {}^{\infty}\!/ \bullet (\textbf{fails } t)* \bullet \textbf{fails } t\ r$

Case $t\ r = \perp$
$= {}^{\infty}\!/ \bullet (\textbf{fails } t) * \perp$ [Definition of **fails**]
$= {}^{\infty}\!/ \perp$ [Definition of $*$]
$= \perp$ [Definition of ${}^{\infty}\!/]$
$= \textbf{fails } t\ r$ [Definition of **fails**]

Case $t\ r = [\,]$
$= {}^{\infty}\!/ \bullet (\textbf{fails } t) * [r]$ [Definition of **fails**]
                    [Definition of **skip**]
$= {}^{\infty}\!/[\textbf{fails } t\ r]$ [Definition of $*$]

145

$= [r]$ [Definition of **fails**]

[Definition of **skip**, $^{\infty}\!/]$

$= $ **fails** $t\ r$ [Definition of $^{\infty}\!/]$

Case $t\ r \neq [\,]$

$= {}^{\infty}\!/ \bullet ($**fails** $t) * [\,]$ [Definition of **fails**]

[Definition of **fail**]

$= [\,]$ [Definition of $*$, $^{\infty}\!/]$

$= $ **fails** $t\ r$ [Definition of **fails**]

[Definition of **fail**]

$\square$

Law 49.   **succs** t; **succs** $t = $ **succs** t

succs t; **succs** $t\ r$

$= {}^{\infty}\!/ \bullet ($**succs** $t)* \bullet $ **succs** $t\ r$

Case $t\ r = \bot$

$= {}^{\infty}\!/ \bullet ($**succs** $t) * \bot$ [Definition of **succs**]

$= \bot$ [Definition of $*$, $^{\infty}\!/]$

$= $ **succs** $t\ r$ [Definition of **succs**]

Case $t\ r = [\,]$

$= {}^{\infty}\!/ \bullet ($**succs** $t) * [\,]$ [Definition of **succs**]

[Definition of **fail**]

$= [\,]$ [Definition of $*$, $^{\infty}\!/]$

$= $ **succs** $t\ r$ [Definition of **succs**]

[Definition of **fail**]

Case $t\ r \neq [\,]$

$= {}^{\infty}\!/ \bullet ($**fails** $t) * [r]$ [Definition of **succs**]

[Definition of **skip**]

$= {}^{\infty}\!/ \bullet [succ\ t\ r]$ [Definition of $*$]

$= [r]$ [Definition of **succs**]

[Definition of **skip**, $^{\infty}\!/]$

$= $ **succs** $t\ r$ [Definition of **skip**]

[Definition of **succs**]

$\square$

146

Law 50^*. **succs** t; **fails** $t =$ **fails** t; **succs** $t =$ **fail**

succs t; **fails** $t\ r$
$= {}^{\bowtie}\!/ \bullet (\textbf{fails }\ t)* \bullet \textbf{succs }\ t\ r$ $\qquad$ [Definition of ;]

Case $t\ r = [\,]$
$= {}^{\bowtie}\!/ \bullet (\textbf{fails }\ t) * [\,]$ $\qquad$ [Definition of **succs**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **fail**]
$= [\,]$ $\qquad$ [Definition of $*, {}^{\bowtie}\!/$]
$= \textbf{fail }\ r$ $\qquad$ [Definition of **fail**]
$= [\,]$
$= {}^{\bowtie}\!/ \bullet [[\,]]$ $\qquad$ [Definition of ${}^{\bowtie}\!/$]
$= {}^{\bowtie}\!/ \bullet [\textbf{succs }\ t\ r]$ $\qquad$ [Definition of **succs**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **fail**]
$= {}^{\bowtie}\!/ \bullet (\textbf{succs }\ t) * [r]$ $\qquad$ [Definition of $*$]
$= {}^{\bowtie}\!/ \bullet (\textbf{succs }\ t)* \bullet \textbf{fails }\ t\ r$ $\qquad$ [Definition of **fails**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **skip**]
$= \textbf{fails }\ t$; **succs** $t\ r$ $\qquad$ [Definition of ;]

Case $t\ r \neq [\,]$
$= {}^{\bowtie}\!/ \bullet (\textbf{fails }\ t) * [r]$ $\qquad$ [Definition of **succs**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **skip**]
$= {}^{\bowtie}\!/[\textbf{fails }\ t\ r]$ $\qquad$ [Definition of $*$]
$= {}^{\bowtie}\!/[[\,]]$ $\qquad$ [Definition of **fails**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **fail**]
$= [\,]$ $\qquad$ [Definition of $*, {}^{\bowtie}\!/$]
$= \textbf{fail }\ r$ $\qquad$ [Definition of **fail**]
$= [\,]$
$= {}^{\bowtie}\!/ \bullet (\textbf{succs }\ t) * [\,]$ $\qquad$ [Definition of ${}^{\bowtie}\!/, *$]
$= {}^{\bowtie}\!/ \bullet (\textbf{succs }\ t)* \bullet \textbf{fails }\ t\ r$ $\qquad$ [Definition of **fails**]
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of **fail**]
$= \textbf{fails }\ t$; **succs** $t\ r$ $\qquad$ [Definition of ;]

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Law 51^*. **fails** $t \mid$ **succs** $t =$ **succs** $t \mid$ **fails** $t =$ **skip**

fails $t \mid$ **succs** $t\ r$
$= {}^{\bowtie}\!/ \bullet < \textbf{fails }\ t\ r, \textbf{succs }\ t\ r >$ $\qquad$ [Definition of $\mid, {}^{\circ}$]

Case $t\ r = [\,]$

$= \bowtie\!/[[r],[\,]]$ [Definition of **fails**]

       [Definition of **succs**]

       [Definition of **skip**]

       [Definition of **fail**]

$= \bowtie\!/[[\,],[r]]$ [Definition of $\bowtie\!/$]

$= [r]$ [Definition of $\bowtie\!/$]

$= \mathbf{skip}\ r$ [Definition of **skip**]

$= [r]$ [Definition of **skip**]

$= \bowtie\!/[[\,],[r]]$ [Definition of $\bowtie\!/$]

$= \bowtie\!/[\mathbf{succs}\ t\ r, \mathbf{fails}\ t\ r]$ [Definition of **succs**]

       [Definition of **fails**]

       [Definition of **skip**]

       [Definition of **fail**]

$= \mathbf{succs}\ t \mid \mathbf{fails}\ t\ r$ [Definition of $^\circ, \bowtie\!/$]

Case $t\ r \neq [\,]$

$= \bowtie\!/[[\,],[r]]$ [Definition of **fails**]

       [Definition of **succs**]

       [Definition of **skip**]

       [Definition of **fail**]

$= \bowtie\!/[[r],[\,]]$ [Definition of $\bowtie\!/$]

$= [r]$ [Definition of $\bowtie\!/$]

$= \mathbf{skip}\ r$ [Definition of **skip**]

$= [r]$ [Definition of **skip**]

$= \bowtie\!/[[r],[\,]]$ [Definition of $\bowtie\!/$]

$= \bowtie\!/[\mathbf{succs}\ t\ r, \mathbf{fails}\ t\ r]$ [Definition of **succs**]

       [Definition of **fails**]

       [Definition of **skip**]

       [Definition of **fail**]

$= \mathbf{succs}\ t \mid \mathbf{fails}\ t\ r$ [Definition of $^\circ, \bowtie\!/$]

$\square$

Law 52. $\mathbf{succs}(t \mid u) = \mathbf{succs}\ t \mid (\mathbf{fails}\ t; \mathbf{succs}\ u)$

$\mathbf{succs}(t \mid u)\ r$

Case $t\ r = \bot$ or $u\ r = \bot$

$= \bot$ [Definition of $\mid, ^\circ, \bowtie\!/$]

       [Definition of **succs**]

148

$= \mathbf{succs}\ t \mid (\mathbf{fails}\ t;\ \mathbf{succs}\ u)\ r$ [Definition of $\mid, \circ, \otimes/$]

[Definition of $\mathbf{succs}$]

[Definition of $\mathbf{fails}$]

Otherwise

$= (\mathbf{fail} \lhd \otimes/\ \bullet < t, u >^\circ\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $\mathbf{succs}, \mid\mid$]

$= (\mathbf{fail} \lhd \otimes/\ \bullet\ t\ r\ \otimes\ u\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $\circ, \otimes/$]

Case $t\ r = [\,],\ u\ r = [\,]$

$= <>$ [Definition of $\lhd\ \rhd$]

[Definition of $\mathbf{fail}$]

$= \otimes/[[\,],[\,]]$ [Definition of $\otimes/$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/[[\,]]]$ [Definition of $\mathbf{succs}$]

[Definition of $\mathbf{fail}, \otimes/$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/ < \mathbf{succs}\ u\ r]]$ [Definition of $\mathbf{succs}$]

[Definition of $\mathbf{fail}$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/\ \bullet\ (\mathbf{succs}\ u) * [r]]$ [Definition of $*$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/\ \bullet\ (\mathbf{succs}\ u) * \bullet\ \mathbf{fails}\ t\ r]$ [Definition of $\mathbf{fails}$]

[Definition of $\mathbf{skip}$]

$= \mathbf{succs}\ t \mid (\mathbf{fails}\ t;\ \mathbf{succs}\ u)\ r$ [Definition of $\circ, \mid\mid$]

Case $t\ r = [\,],\ u\ r \neq [\,]$

$= [r]$ [Definition of $\lhd\ \rhd$]

[Definition of $\mathbf{skip}$]

$= \otimes/[[\,],[r]]$ [Definition of $\otimes/$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/[[r]]]$ [Definition of $\mathbf{succs}$]

[Definition of $\mathbf{fail}, \otimes/$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/[\mathbf{succs}\ u\ r]]$ [Definition of $\mathbf{succs}$]

[Definition of $\mathbf{skip}$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/\ \bullet\ (\mathbf{succs}\ u) * [r]]$ [Definition of $*$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/\ \bullet\ (\mathbf{succs}\ u) * \bullet\ \mathbf{fails}\ t\ r]$ [Definition of $\mathbf{fails}$]

[Definition of $\mathbf{skip}$]

$= \mathbf{succs}\ t \mid (\mathbf{fails}\ t;\ \mathbf{succs}\ u)\ r$ [Definition of $\circ, \mid\mid$]

Case $t\ r \neq [\,]$

$= [r]$ [Definition of $\lhd\ \rhd$]

[Definition of $\mathbf{skip}$]

$= \otimes/[[r],[\,]]$ [Definition of $\otimes/$]

$= \otimes/[\mathbf{succs}\ t\ r, \otimes/\ \bullet\ (\mathbf{succs}\ u) * [\,]]$ [Definition of $\mathbf{succs}$]

[Definition of $\mathbf{skip}, *, \otimes/$]

149

$$= {}^{\otimes}\!/[\mathbf{succs}\ t\ r, {}^{\otimes}\!/\bullet(\mathbf{succs}\ u)*\bullet\mathbf{fails}\ t\ r] \qquad\qquad [\text{Definition of }\mathbf{fails}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Definition of }\mathbf{fail}]$$
$$= \mathbf{succs}\ t\mid(\mathbf{fails}\ t;\ \mathbf{succs}\ u)\ r \qquad\qquad\qquad\qquad [\text{Definition of }^{\circ},\|]$$

$\square$

Law 53. $\mathbf{succs}(\mathbf{fails}\ s;\ t)) = \mathbf{fails}\ s;\ \mathbf{succs}\ t$

$\mathbf{succs}(\mathbf{fails}\ s;\ t))\ r$

Case $s\ r = \bot$ or $t\ r = \bot$
$$= \bot \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Definition of }\mathbf{fails},;\ ]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{succs}]$$
$$= \mathbf{fails}\ s;\ \mathbf{succs}\ t\ r \qquad\qquad\qquad\qquad\qquad\ [\text{Definition of };\ ,\mathbf{succs}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{fails}]$$

Case $s\ r = [\,]$, $t\ r = [\,]$
$$= (\mathbf{fail}\lhd {}^{\otimes}\!/\bullet t*[r] = [\,]\rhd\mathbf{skip})\ r \qquad\qquad [\text{Definition of }\mathbf{fails}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{skip}]$$
$$= (\mathbf{fail}\lhd t\ r = [\,]\rhd\mathbf{skip})\ r \qquad\qquad\qquad [\text{Definition of }*, {}^{\otimes}\!/]$$
$$= [\,] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{Definition of }\lhd\rhd]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{fail}]$$
$$= {}^{\otimes}\!/[[\,]] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of } {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\mathbf{succs}\ t\ r] \qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{succs}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{fail}]$$
$$= {}^{\otimes}\!/(\mathbf{succs}\ t)*[r] \qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }*]$$
$$= {}^{\otimes}\!/(\mathbf{succs}\ t)*\bullet\mathbf{fails}\ s\ r \qquad\qquad\qquad [\text{Definition of }\mathbf{fails}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{skip}]$$
$$= \mathbf{fails}\ s;\ \mathbf{succs}\ t\ r \qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of };\ ]$$

Case $s\ r = [\,]$, $t\ r \neq [\,]$
$$= (\mathbf{fail}\lhd {}^{\otimes}\!/\bullet t*[r] = [\,]\rhd\mathbf{skip})\ r \qquad\qquad [\text{Definition of }\mathbf{fails}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{skip}]$$
$$= (\mathbf{fail}\lhd t\ r = [\,]\rhd\mathbf{skip})\ r \qquad\qquad\qquad [\text{Definition of }*, {}^{\otimes}\!/]$$
$$= [r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\lhd\rhd]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{skip}]$$
$$= {}^{\otimes}\!/[[r]] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of } {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\mathbf{succs}\ t\ r] \qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{succs}]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }\mathbf{skip}]$$
$$= {}^{\otimes}\!/(\mathbf{succs}\ t)*[r] \qquad\qquad\qquad\qquad\qquad\quad [\text{Definition of }*]$$

150

$= \, \% /(\text{succs} \ t) * \bullet \text{fails} \ s \ r$ [Definition of **fails**]

                                                [Definition of **skip**]

$= \text{fails} \ s; \ \text{succs} \ t \ r$ [Definition of ;]

Case $s \ r \neq [\,]$

$= (\textbf{fail} \lhd \, \% / \bullet t * [\,] = [\,] \rhd \textbf{skip}) \ r$ [Definition of **fails**]

                                                [Definition of **fail**]

$= [\,]$ [Definition of $*, \, \% /$]

                                                [Definition of **fail**]

$= \, \% /(\text{succs} \ t) * [\,]$ [Definition of $*$]

$= \, \% /(\text{succs} \ t) * \bullet \text{fails} \ s \ r$ [Definition of **fails**]

                                                [Definition of **fail**]

$= \text{fails} \ s; \ \text{succs} \ t \ r$ [Definition of ;]

                                               □

Law 54. $\textbf{succs}(\textbf{succs} \ s; \ t) = \textbf{succs} \ s; \ \textbf{succs} \ t$

$\textbf{succs}(\textbf{succs} \ s; \ t)) \ r$

Case $s \ r = \bot$ or $t \ r = \bot$

$= \bot$ [Definition of **succs**,;]

$= \textbf{succs} \ s; \ \textbf{succs} \ t \ r$ [Definition of **succs**,;]

Otherwise

$= (\textbf{fail} \lhd \textbf{succs} \ s; \ t \ r = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]

$= (\textbf{fail} \lhd \, \% / \bullet t * \bullet \textbf{succs} \ s \ r = [\,] \rhd \textbf{skip}) \ r$ [Definition of ;]

Case $s \ r = [\,]$

$= (\textbf{fail} \lhd \, \% / \bullet t * [\,] = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]

                                                [Definition of **fail**]

$= [\,]$ [Definition of $*, \, \% /$]

                                                [Definition of $\lhd \rhd, \textbf{fail}$]

$= \, \% /(\text{succs} \ t) * [\,]$ [Definition of $*$]

$= \, \% /(\text{succs} \ t) * \bullet \textbf{succs} \ s \ r$ [Definition of **succs**]

                                                [Definition of **fail**]

$= \textbf{succs} \ s; \ \textbf{succs} \ t \ r$ [Definition of ;]

Case $s \ r \neq [\,], t \ r = [\,]$

$= (\textbf{fail} \lhd \, \% / \bullet t * [r] = [\,] \rhd \textbf{skip}) \ r$ [Definition of **succs**]

                                                [Definition of **skip**]

$= (\textbf{fail} \lhd t\ r = [\ ] \rhd \textbf{skip})\ r$ [Definition of $*$, $\bowtie\!/$]

$= [\ ]$ [Definition of $\lhd\ \rhd$]

     [Definition of $\textbf{fail}$]

$= \bowtie\!/[[\ ]]$ [Definition of $\bowtie\!/$]

$= \bowtie\!/[\textbf{succs}\ t\ r]$ [Definition of $\textbf{succs}$]

     [Definition of $\textbf{fail}$]

$= \bowtie\!/(\textbf{succs}\ t) * [r]$ [Definition of $*$]

$= \bowtie\!/(\textbf{succs}\ t) * \bullet \textbf{succs}\ s\ r$ [Definition of $\textbf{succs}$]

     [Definition of $\textbf{skip}$]

$= \textbf{succs}\ s; \textbf{succs}\ t\ r$ [Definition of $;$]

Case $s\ r \neq [\ ], t\ r \neq [\ ]$

$= (\textbf{fail} \lhd \bowtie\!/ \bullet t * [r] = [\ ] \rhd \textbf{skip})\ r$ [Definition of $\textbf{succs}$]

     [Definition of $\textbf{skip}$]

$= (\textbf{fail} \lhd t\ r = [\ ] \rhd \textbf{skip})\ r$ [Definition of $*$, $\bowtie\!/$]

$= [r]$ [Definition of $\lhd\ \rhd$]

     [Definition of $\textbf{skip}$]

$= \bowtie\!/[[r]]$ [Definition of $\bowtie\!/$]

$= \bowtie\!/[\textbf{succs}\ t\ r]$ [Definition of $\textbf{succs}$]

     [Definition of $\textbf{skip}$]

$= \bowtie\!/(\textbf{succs}\ t) * [r]$ [Definition of $*$]

$= \bowtie\!/(\textbf{succs}\ t) * \bullet \textbf{succs}\ s\ r$ [Definition of $\textbf{succs}$]

     [Definition of $\textbf{skip}$]

$= \textbf{succs}\ s; \textbf{succs}\ t\ r$ [Definition of $;$]

$\square$

Law 55. $\textbf{succs}(s; \textbf{fails}\ t) = \textbf{succs}\ s; \textbf{fails}(s;\ t)$

$\textbf{succs}(s; \textbf{fails}\ t)\ r$

Case $s\ r = \bot$ or $t\ r = \bot$

$= \bot$ [Definition of $\textbf{succs}$,;]

$= \textbf{succs}\ s; \textbf{fails}(s;\ t)\ r$ [Definition of $\textbf{succs}$,;]

Otherwise

$= (\textbf{fail} \lhd \bowtie\!/ \bullet (\textbf{fails}\ t) * \bullet s\ r = <> \rhd \textbf{skip})\ r$ [Definition of $\textbf{succs}$, ;]

Case $s\ r = <>$

$= (\textbf{fail} \lhd <> = <> \rhd \textbf{skip})\ r$ [Definition of $*$, $\bowtie\!/$]

152

$= <>$ [Definition of $\lhd \rhd$]

[Definition of **fail**]

$= \text{\reflectbox{$\propto$}}/ \bullet (\textbf{fails } (s; \ t))* \ <>$ [Definition of $\text{\reflectbox{$\propto$}}/, *$]

$= \text{\reflectbox{$\propto$}}/ \bullet (\textbf{fails } (s; \ t))* \bullet \textbf{succs } s \ r$ [Definition of **succs**]

[Definition of **fail**]

$= \textbf{succs } s; \ \textbf{fails}(s; \ t) \ r$ [Definition of ;]

Case $s \ r \neq <>$

$= (\textbf{skip} \lhd \text{\reflectbox{$\propto$}}/ \bullet (\textbf{succs } t)* \bullet s \ r =<> \rhd \textbf{fail}) \ r$

$= (\textbf{skip} \lhd \text{\reflectbox{$\propto$}}/ \bullet t* \bullet s \ r =<> \rhd \textbf{fail}) \ r$ [Law 42]

$= \textbf{fails } (s; \ t) \ r$ [Definition of ;]

$= \text{\reflectbox{$\propto$}}/ \bullet (\textbf{fails } (s; \ t))* \ < r >$ [Definition of $*$]

$= \text{\reflectbox{$\propto$}}/ \bullet (\textbf{fails } (s; \ t))* \bullet \textbf{succs } s \ r$ [Definition of **succs**]

[Definition of **skip**]

$= \textbf{succs } s; \ \textbf{fails}(s; \ t) \ r$ [Definition of ;]

□

Law 56. $\textbf{succs}(s; \ \textbf{succs } t) = \textbf{succs } s; \ \textbf{succs } (s; \ t)$

$\textbf{succs}(s; \ \textbf{succs } t)$

$= \textbf{succs } (s; \ t)$ [Law 42]

$= \textbf{succs } s; \ \textbf{succs } (s; \ t)$ [Law 29]

□

Law 57. $\textbf{fails } (\textbf{succs } s; \ t) = \textbf{fails } s \mid (\textbf{succs } s; \ \textbf{fails } t)$

$\textbf{fails } (\textbf{succs } s; \ t) \ r$

Case $s \ r = \perp$ ot $t \ r = \perp$

$= \perp$ [Definition of ; ,**succs**]

[Definition of **fails**]

$= \textbf{fails } s \mid (\textbf{succs } s; \ \textbf{fails } t) \ r$ [Definition of $\mid$,; ,**fails**]

[Definition of **succs**]

Otherwise

$= \textbf{fails } (\textbf{succs } (\textbf{succs } s); \ \textbf{succs } t) \ r$ [Law 35]

$= \textbf{fails } (\textbf{succs } s; \ \textbf{succs } t) \ r$ [Law 40]

$= (\textbf{skip} \lhd \text{\reflectbox{$\propto$}}/ \bullet (\textbf{succs } t)* \bullet \textbf{succs } s \ r = [\ ] \rhd \textbf{fail}) \ r$ [Definition of **succs**, ;]

153

Case $s\ r = [\,]$

$= (\textbf{skip} \lhd \otimes\!/ \bullet (\textbf{succs}\ t) * [\,] = [\,] \rhd \textbf{fail})\ r$ [Definition of **succs**]

  [Definition of **fail**]

$= [r]$ [Definition of $*, \otimes\!/$]

  [Definition of $\lhd\ \rhd, \textbf{skip}$]

$= \otimes\!/[[r],[\,]]$ [Definition of $\otimes\!/$]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/ \bullet (\textbf{fails}\ t) * [\,]]$ [Definition of **fails**]

  [Definition of $\textbf{skip}, \otimes\!/, *$]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/ \bullet (\textbf{fails}\ t) * \bullet \textbf{succs}\ s\ r]$ [Definition of **succs**]

  [Definition of **fail**]

$= \textbf{fails}\ s\ |\ (\textbf{succs}\ s;\ \textbf{fails}\ t)\ r$ [Definition of $;\ °,|$]

Case $s\ r \neq [\,], t\ r = [\,]$

$= (\textbf{skip} \lhd \otimes\!/ \bullet (\textbf{succs}\ t) * [r] = [\,] \rhd \textbf{fail})\ r$ [Definition of **succs**]

  [Definition of **skip**]

$= (\textbf{skip} \lhd \textbf{succs}\ t\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of $*, \otimes\!/$]

$= [r]$ [Definition of **succs**]

  [Definition of **fail**]

  [Definition of **skip**]

$= \otimes\!/[[\,],[r]]$ [Definition of $\otimes\!/$]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/[\textbf{fails}\ t\ r]]$ [Definition of **fails**]

  [Definition of **fail**]

  [Definition of **skip**]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/ \bullet (\textbf{fails}\ t) * [r]]$ [Definition of $*$]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/ \bullet (\textbf{fails}\ t) * \bullet \textbf{succs}\ s\ r]$ [Definition of **succs**]

  [Definition of **skip**]

$= \textbf{fails}\ s\ |\ (\textbf{succs}\ s;\ \textbf{fails}\ t)\ r$ [Definition of $;\ °,|$]

Case $s\ r \neq [\,], t\ r \neq [\,]$

$= (\textbf{skip} \lhd \otimes\!/ \bullet (\textbf{succs}\ t) * [r] = [\,] \rhd \textbf{fail})\ r$ [Definition of **succs**]

  [Definition of **skip**]

$= (\textbf{skip} \lhd \textbf{succs}\ t\ r = [\,] \rhd \textbf{fail})\ r$ [Definition of $*, \otimes\!/$]

$= [\,]$ [Definition of **succs**]

  [Definition of **skip**]

  [Definition of $\lhd\ \rhd$]

  [Definition of **fail**]

$= \otimes\!/[[\,],[\,]]$ [Definition of $\otimes\!/$]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/[\textbf{fails}\ t\ r]$ [Definition of $\otimes\!/, \textbf{fails}$]

  [Definition of **fail**]

$= \otimes\!/[\textbf{fails}\ s\ r, \otimes\!/ \bullet (\textbf{fails}\ t) * [r]]$ [Definition of $*$]

$$= {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } t)\ast \bullet \textbf{succs } s \ r] \qquad \text{[Definition of } \textbf{succs}]$$
$$\text{[Definition of } \textbf{skip}]$$
$$= \textbf{fails } s \mid (\textbf{succs } s;\ \textbf{fails } t) \ r \qquad \text{[Definition of } ;\ {}^{\circ}, |]$$

$\square$

Law 58. $\textbf{fails}(s;\ \textbf{succs } t) = \textbf{fails } s \mid (\textbf{succs } s;\ \textbf{fails } (s;\ t))$
This law is proved by induction, as follows:

$\textbf{fails}(s;\ \textbf{succs } t) \ r$

Base Case: $s \ r = \bot$
$$= \bot \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } \textbf{fails}]$$
$$= {}^{\otimes}\!/\ \bot \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\bot, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t)) \ast [\,]] \qquad\qquad \text{[Definition of } {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t))\ast \bullet \textbf{succs } s \ r > \qquad \text{[Definition of } \textbf{succs}]$$
$$\text{[Definition of } \textbf{fails}]$$
$$= \textbf{fails } s \mid (\textbf{succs } s;\ \textbf{fails } (s;\ t)) \ r \qquad \text{[Definition of } ;\ ,{}^{\circ}, |]$$

Base Case: $s \ r = [\,]$
$$= (\textbf{skip} \lhd {}^{\otimes}\!/ \bullet (\textbf{succs } t)\ast \bullet s \ r = [\,] \rhd \textbf{fail}) \ r \qquad \text{[Definition of } \textbf{fails}, ;\ ]$$
$$= (\textbf{skip} \lhd [\,] = [\,] \rhd \textbf{fail}) \ r \qquad\qquad\qquad \text{[Definition of } \ast, {}^{\otimes}\!/]$$
$$= [r] \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } \lhd \rhd]$$
$$\text{[Definition of } \textbf{skip}]$$
$$= {}^{\otimes}\!/[[r], [\,]] \qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t)) \ast [\,]] \qquad \text{[Definition of } \textbf{fails}]$$
$$\text{[Definition of } \textbf{skip}, \ast, {}^{\otimes}\!/]$$
$$= {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t))\ast \bullet \textbf{succs } s \ r] \qquad \text{[Definition of } \textbf{succs}]$$
$$\text{[Definition of } \textbf{fail}]$$
$$= \textbf{fails } s \mid (\textbf{succs } s;\ \textbf{fails } (s;\ t)) \ r \qquad \text{[Definition of } ;\ ,{}^{\circ}, |]$$

Inductive step on $s \ r$:
We suppose
$(\textbf{skip} \lhd {}^{\otimes}\!/ \bullet (\textbf{succs } t) \ast xs =$
$\qquad [\,] \rhd \textbf{fail}) \ r = {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t))\ast \bullet \textbf{succs } s \ r]$
We must prove that
$(\textbf{skip} \lhd {}^{\otimes}\!/ \bullet (\textbf{succs } t) \ast [x] {}^{\otimes} xs =$
$\qquad [\,] \rhd \textbf{fail}) \ r = {}^{\otimes}\!/[\textbf{fails } s \ r, {}^{\otimes}\!/ \bullet (\textbf{fails } (s;\ t))\ast \bullet \textbf{succs } s \ r]$

$(\textbf{skip} \lhd {}^{\otimes}\!/ \bullet (\textbf{succs } t) \ast [x] {}^{\otimes} xs = [\,] \rhd \textbf{fail}) \ r$
$= (\textbf{skip} \lhd {}^{\otimes}\!/ \bullet ((\textbf{succs } t \ x) {}^{\otimes} (\textbf{succs } t) \ast xs = [\,] \rhd \textbf{fail}) \ r \quad \text{[Definition of } \ast]$

155

Case $t \; x = [\,]$
$= (\textbf{skip} \lhd \bowtie\!/ \bullet ([\,] \bowtie (\textbf{succs} \; t) * \; xs = [\,] \rhd \textbf{fail}) \; r$ [Assumption]
$= (\textbf{skip} \lhd \bowtie\!/ \bullet (\textbf{succs} \; t) * \; xs = [\,] \rhd \textbf{fail}) \; r$ [Definition of $\bowtie$]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (s;\; t)) * \bullet \; \textbf{succs} \; s \; r]$ [Assumption]

Case $t \; x = \bot$
$= \bot$ [Definition of **succs**]
     [Definition of **fails**]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (s;\; t)) * \bullet \; \textbf{succs} \; s \; r]$ [Definition **succs**]
     [Definition of **fails**, $\bowtie\!/]$

Case $t \; x = xs$
$= (\textbf{skip} \lhd \bowtie\!/ \bullet ([x] \bowtie (\textbf{succs} \; t) * \; xs = [\,] \rhd \textbf{fail}) \; r$ [Assumption]
$= (\textbf{skip} \lhd [x] \bowtie \bowtie\!/ \bullet (\textbf{succs} \; t) * \; xs = [\,] \rhd \textbf{fail}) \; r$ [Definition of $\bowtie\!/]$
$= [r]$ [Definition of $\lhd \rhd$]
$= \bowtie\!/[[\,], [r]]$ [Definition of $\bowtie\!/]$
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet \textbf{fails} \; (\bowtie\!/ \bullet t * \bullet \; s) \; r]$ [Assumption]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (\bowtie\!/ \bullet t * \bullet \; s)) * [r]]$ [Definition $*$]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (s;\; t)) * [r]]$ [Definition of ;]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (s;\; t)) * \bullet \; \textbf{skip} \; r]$ [Definition **skip**]
$= \bowtie\!/[\textbf{fails} \; s \; r, \bowtie\!/ \bullet (\textbf{fails} \; (s;\; t)) * \bullet \; \textbf{succs} \; s \; r]$ [Definition **succs**]

$\square$

D.4   Laws involving Structural Combinators

Law 59. $(t_1 \;\boxed{;}\; t_2); \; (t_3 \;\boxed{;}\; t_4) = (t_1; \; t_3) \;\boxed{;}\; (t_2; \; t_4)$

$(\llbracket (t_1 \boxed{;} t_2); \; (t_3 \boxed{;} t_4) \rrbracket \Gamma_L \; \Gamma_T)(p_1; \; p_2, proofs)$
$= \bowtie\!/ \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L \; \Gamma_T) * \bullet \; \llbracket (t_1 \boxed{;} t_2) \rrbracket \Gamma_L \; \Gamma_T \; (p_1; \; p_2, proofs)$ [Definition of ;]
$= \bowtie\!/ \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L \; \Gamma_T) * \bullet$
   $\Omega_{;} * (\Pi(< \llbracket t_1 \rrbracket \Gamma_L \; \Gamma_T \; (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L \; \Gamma_T \; (p_2, proofs) >))$ [Definition of $\boxed{;}$]
$= \bowtie\!/ \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L \; \Gamma_T) * \bullet$
   $\Omega_{;} * [(p_1', pr_1') : as \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \; \Gamma_T \; (p_1, proofs),$
                   $as \leftarrow \Pi < \llbracket t_2 \rrbracket \Gamma_L \; \Gamma_T \; (p_2, proofs) >]$ [Definition of Π]
$= \bowtie\!/ \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L \; \Gamma_T) * \bullet$
   $\Omega_{;} * [(p_1', pr_1') : as \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \; \Gamma_T \; (p_1, proofs),$
                   $as \leftarrow e2l * < \llbracket t_2 \rrbracket \Gamma_L \; \Gamma_T \; (p_2, proofs)]]$ [Definition of Π]

$$= \bowtie\!/ \bullet ([\![(t_3\,\boxed{;}\,t_4)]\!]\Gamma_L\ \Gamma_T) * \bullet$$
$$\Omega_;\ *\,[< (p_1', pr_1'), (p_2', pr_2') >$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)] \qquad \text{[Lemma D.6.8]}$$
$$= \bowtie\!/ \bullet ([\![(t_3\,\boxed{;}\,t_4)]\!]\Gamma_L\ \Gamma_T) *$$
$$[\Omega_;\ < (p_1', pr_1'), (p_2', pr_2') >$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)] \qquad \text{[Definition of $*$]}$$
$$= \bowtie\!/ \bullet ([\![(t_3\,\boxed{;}\,t_4)]\!]\Gamma_L\ \Gamma_T) *$$
$$[(p_1';\ p_2', pr_1' \cup pr_2') \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)] \qquad \text{[Definition of $\Omega_;$]}$$
$$= \bowtie\!/[[\![t_3\,\boxed{;}\,t_4]\!]\Gamma_L\ \Gamma_T(p_1';\ p_2', pr_1' \cup pr_2')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)] \qquad \text{[Definition of $*$]}$$
$$= \bowtie\!/[\Omega_;\ *\,(\Pi < [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') >)$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)] \qquad \text{[Definition of $\boxed{;}$]}$$
$$= \bowtie\!/[\Omega_;\ *\,[(p_3', pr_3') : cs$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$cs \leftarrow \Pi < [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')]] \qquad \text{[Definition of Π]}$$
$$= \bowtie\!/[\Omega_;\ *\,[(p_3', pr_3') : cs$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$cs \leftarrow e2l * [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')]] \qquad \text{[Definition of Π]}$$
$$= \bowtie\!/[\Omega_;\ *\,[< (p_3', pr_3'), (p_4', pr_4') >$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$(p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')]] \qquad \text{[Lemma D.6.8]}$$
$$= \bowtie\!/[\Omega_;\ < (p_3', pr_3'), (p_4', pr_4') >$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$(p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')] \qquad \text{[Definition of $*$]}$$

$$= \bowtie/[(p_3';\ p_4', pr_3' \cup pr_4')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$(p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')] \qquad \text{[Definition of } \Omega_; \text{]}$$
$$= [(p_3';\ p_4', pr_3' \cup pr_4')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'),$$
$$(p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2')] \qquad \text{[Definition of } \bowtie/\text{]}$$
$$= [(p_3';\ p_4', pr_3' \cup pr_4')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3') \leftarrow (add\ pr_2') * [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'),$$
$$(p_4', pr_4') \leftarrow (add\ pr_1') * [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2')] \qquad \text{[Definition of } add\text{]}$$
$$= [(p_3';\ p_4', pr_3' \cup pr_4')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'),$$
$$(p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2'),$$
$$pr_3' = pr_3'' \cup pr_2', pr_4' = pr_4'' \cup pr_1'] \qquad \text{[Property of } add\text{]}$$
$$= [(p_3';\ p_4', pr_3'' \cup pr_4'' \cup pr_2' \cup pr_1')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs),$$
$$(p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'),$$
$$(p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2'),$$
$$pr_3' = pr_3'' \cup pr_2', pr_4' = pr_4'' \cup pr_1'] \qquad \text{[Property of } \cup\text{]}$$
$$= [(p_3';\ p_4', pr_3'' \cup pr_4'')$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)$$
$$(p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'),$$
$$(p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2')] \qquad \text{[Lemma D.6.9]}$$
$$= [\Omega_;\ < (p_3', pr_3''), (p_4', pr_4'') >$$
$$\mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs),$$
$$(p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs)$$
$$(p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'),$$
$$(p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2')] \qquad \text{[Definition of } \Omega_; \text{]}$$

$$= \Omega_; \; * [< (p_3', pr_3''), (p_4', pr_4'') >$$
$$\qquad | \; (p_1', pr_1') \leftarrow [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad \quad (p_2', pr_2') \leftarrow [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs)$$
$$\qquad \quad (p_3', pr_3'') \leftarrow [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_1', pr_1'),$$
$$\qquad \quad (p_4', pr_4'') \leftarrow [\![t_4]\!] \Gamma_L \; \Gamma_T \; (p_2', pr_2')] \qquad\qquad\qquad \text{[Definition of } *]$$
$$= \Omega_; \; * (\Pi < [(p_3', pr_3'')$$
$$\qquad\qquad | \; (p_1', pr_1') \leftarrow [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad \quad (p_3', pr_3'') \leftarrow [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_1', pr_1')],$$
$$\qquad\qquad [(p_4', pr_4'')$$
$$\qquad\qquad | \; (p_2', pr_2') \leftarrow [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs),$$
$$\qquad\qquad \quad (p_4', pr_4'') \leftarrow [\![t_4]\!] \Gamma_L \; \Gamma_T \; (p_2', pr_2')] >) \qquad\qquad \text{[Definition of } \Pi]$$
$$= \Omega_; \; * (\Pi < {}^{⋈}\!/[(p_3', pr_3'')$$
$$\qquad\qquad\quad | \; (p_1', pr_1') \leftarrow [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad\quad \quad (p_3', pr_3'') \leftarrow [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_1', pr_1')],$$
$$\qquad\qquad {}^{⋈}\!/[(p_4', pr_4'')$$
$$\qquad\qquad\quad | \; (p_2', pr_2') \leftarrow [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs),$$
$$\qquad\qquad\quad \quad (p_4', pr_4'') \leftarrow [\![t_4]\!] \Gamma_L \; \Gamma_T \; (p_2', pr_2')] >) \qquad \text{[Definition of } {}^{⋈}\!/]$$
$$= \Omega_; \; * (\Pi[{}^{⋈}\!/ \bullet [\![t_3]\!] \Gamma_L \; \Gamma_T * < (p_1', pr_1') \; | \; (p_1', pr_1') \leftarrow [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs)],$$
$$\qquad\qquad {}^{⋈}\!/ \bullet [\![t_4]\!] \Gamma_L \; \Gamma_T * < (p_2', pr_2') \; | \; (p_2', pr_2') \leftarrow [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs)] >)$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Definition of } [\![]\!] \text{ and } *]$$
$$= \Omega_; \; * (\Pi < {}^{⋈}\!/ \bullet [\![t_3]\!] \Gamma_L \; \Gamma_T * \bullet [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad {}^{⋈}\!/ \bullet [\![t_4]\!] \Gamma_L \; \Gamma_T * \bullet [\![t_4]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >) \qquad \text{[Definition of } [\![]\!]]$$
$$= \Omega_; \; * (\Pi < [\![t_1; \; t_3]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad [\![t_2; \; t_4]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >) \qquad\qquad\qquad \text{[Definition of } ; \;]$$
$$= [\![(t_1; \; t_3) \fbox{;} (t_2; \; t_4)]\!] \Gamma_L \; \Gamma_T \; (p_1; \; p_2, proofs) \qquad\qquad \text{[Definition of } \fbox{;}]$$

$$\square$$

Law 60. $t_1 \fbox{;} (t_2 \mid t_3) = (t_1 \fbox{;} t_2) \mid (t_1 \fbox{;} t_3)$

$$[\![t_1 \fbox{;} (t_2 \mid t_3)]\!] \Gamma_L \; \Gamma_T \; (p_1; \; p_2, proofs)$$
$$= \Omega_; \; * (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad\quad [\![t_2 \mid t_3]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >) \qquad\qquad \text{[Definition of } \fbox{;}]$$
$$= \Omega_; \; * (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad\quad {}^{⋈}\!/[[\![t_2]\!] \Gamma_L \; \Gamma_T, [\![t_3]\!] \Gamma_L \; \Gamma_T]^\circ \; (p_2, proofs) >) \qquad \text{[Definition of } \mid]$$
$$= \Omega_; \; * (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs),$$
$$\qquad\qquad\quad ([\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs)) \; {}^{⋈} \; ([\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs)) >) \; \text{[Definition of } \circ]$$
$$= \Omega_; \; * ((\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs), [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >)^{⋈}$$
$$\qquad\quad (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs), [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >)) \qquad \text{[Lemma D.6.10]}$$
$$= \Omega_; \; * (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs), [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >)^{⋈}$$
$$\quad \Omega_; \; * (\Pi < [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p_1, proofs), [\![t_3]\!] \Gamma_L \; \Gamma_T \; (p_2, proofs) >) \qquad \text{[Lemma D.6.12]}$$

159

$$= \,⅋/(\Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)^{⅋}$$
$$\Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)) \quad \text{[Property of } ⅋/]$$
$$= \,⅋/([\![t_1 \boxed{;} t_2]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)^{⅋}$$
$$[\![t_1 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)) \quad \text{[Definition of } \boxed{;}]$$
$$= \,⅋/[[\![t_1 \boxed{;} t_2]\!]\Gamma_L \; \Gamma_T, [\![t_1 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T]^{\circ} \;(p_1; \; p_2, proofs)) \quad \text{[Definition of } \circ]$$
$$= [\![(t_1 \boxed{;} t_2) \mid (t_1 \boxed{;} t_3)]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)) \quad \text{[Definition of } |]$$

$\square$

Law 61. $(t_1 \mid t_2) \boxed{;} t_3 = (t_1 \boxed{;} t_3) \mid (t_2 \boxed{;} t_3)$

$$[\![(t_1 \mid t_2) \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)$$
$$= \Omega_; \,*(\Pi < [\![t_1 \mid t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Definition of } \boxed{;}]$$
$$= \Omega_; \,*(\Pi < \,⅋/[[\![t_1]\!]\Gamma_L \; \Gamma_T, [\![t_2]\!]\Gamma_L \; \Gamma_T]^{\circ} \;(p_1, proofs),$$
$$[\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Definition of } |]$$
$$= \Omega_; \,*(\Pi < \,⅋/([\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs)$$
$$^{⅋}[\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs)), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Definition of } \circ]$$
$$= \Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs)$$
$$^{⅋}[\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Definition of } ⅋/]$$
$$= \Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >^{⅋}$$
$$\Pi < [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Lemma D.6.11]}$$
$$= \Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)^{⅋}$$
$$\Omega_; \,*(\Pi < [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Lemma D.6.12]}$$
$$= \,⅋/(\Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)^{⅋}$$
$$\Omega_; \,*(\Pi < [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_3]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)) \quad \text{[Definition of } ⅋/]$$
$$= \,⅋/([\![t_1 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T(p_1; \; p_2, proofs) \,^{⅋} [\![t_2 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T(p_1; \; p_2, proofs)) \quad \text{[Definition of } \boxed{;}]$$
$$= \,⅋/ \bullet [[\![t_1 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T, [\![t_2 \boxed{;} t_3]\!]\Gamma_L \; \Gamma_T]^{\circ} \;(p_1; \; p_2, proofs)) \quad \text{[Definition of } \circ]$$
$$= [\![(t_1 \boxed{;} t_3) \mid (t_2 \boxed{;} t_3)]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)) \quad \text{[Definition of } |]$$

$\square$

Law 62. $!(t_1 \boxed{;} t_2) = !t_1 \boxed{;} !t_2)$

$$[\![!(t_1 \boxed{;} t_2)]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs)$$
$$= head'(\Omega_; \,*(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)) \;\text{[Definition of ! and } \boxed{;}]$$
$$= \Omega_; \,*(head'(\Pi < [\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >)) \;\text{[Lemma D.6.13]}$$
$$= \Omega_; \,*(\Pi < head'([\![t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs)),$$
$$head'([\![t_2]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs)) >) \quad \text{[Lemma D.6.14]}$$
$$= \Omega_; \,*(\Pi < [\![!t_1]\!]\Gamma_L \; \Gamma_T \;(p_1, proofs), [\![!t_2]\!]\Gamma_L \; \Gamma_T \;(p_2, proofs) >) \quad \text{[Definition of !]}$$
$$= [\![!t_1 \boxed{;} !t_2]\!]\Gamma_L \; \Gamma_T \;(p_1; \; p_2, proofs) \quad \text{[Definition of } \boxed{;}]$$

$\square$

160

D.5 Laws on con

The side-conditions of these laws use the function ϕ which extracts the set of free-variables of the tactic to which it is applied.

Law 63. $(\textbf{con } v \bullet t_1;\ t_2) = (\textbf{con } v \bullet t_1);\ t_2$ provided $v \notin \phi t_2$

$$
\begin{aligned}
&(\llbracket \textbf{con } v \bullet t_1;\ t_2 \rrbracket \Gamma_L\ \Gamma_T) \\
&= (\llbracket (|_{v \in TERM}(t_1;\ t_2)(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}] \\
&= (\llbracket (|_{v \in TERM}(t_1(v);\ t_2(v))) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Distribution of } ;] \\
&= (\llbracket (|_{v \in TERM}\, t_1(v);\ t_2) \rrbracket \Gamma_L\ \Gamma_T)) && [\text{provided } v \notin \phi t_2] \\
&= (\llbracket (|_{v \in TERM}\, t_1(v));\ t_2 \rrbracket \Gamma_L\ \Gamma_T) && [\text{Law 6}] \\
&= (\llbracket (\textbf{con } v \bullet t_1);\ t_2 \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}]
\end{aligned}
$$

□

Law 64. $(\textbf{con } v \bullet t_1;\ t_2) = t_1;\ (\textbf{con } v \bullet t_2)$ provided $v \notin \phi t_1$ and $!t_1 = t_1$

$$
\begin{aligned}
&(\llbracket (\textbf{con } v \bullet t_1;\ t_2) \rrbracket \Gamma_L\ \Gamma_T) \\
&= (\llbracket (|_{v \in TERM}(t_1;\ t_2)(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}] \\
&= (\llbracket (|_{v \in TERM}(t_1(v);\ t_2(v))) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Distribution of } ;] \\
&= (\llbracket (|_{v \in TERM}\, t_1;\ t_2(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{provided } v \notin \phi t_2] \\
&= (\llbracket t_1;\ (|_{v \in TERM}\, t_2(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Law 7}] \\
&= (\llbracket t_1;\ (\textbf{con } v \bullet t_2) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}]
\end{aligned}
$$

□

Law 65. $(\textbf{con } v \bullet t) = t$ provided $v \notin \phi t$

$$
\begin{aligned}
&(\llbracket (\textbf{con } v \bullet t) \rrbracket \Gamma_L\ \Gamma_T) \\
&= (\llbracket (|_{v \in TERM}\, t(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}] \\
&= (\llbracket (|_{v \in TERM}\, t) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Provided } v \notin \phi t] \\
&= (\llbracket t \rrbracket \Gamma_L\ \Gamma_T) && [t \mid t = t]
\end{aligned}
$$

□

Law 66. $(\textbf{con } v \bullet t) = (\textbf{con } u \bullet t[v \setminus u])$ provided $u \notin \phi t$

$$
\begin{aligned}
&(\llbracket (\textbf{con } v \bullet t) \rrbracket \Gamma_L\ \Gamma_T) \\
&= (\llbracket (|_{v \in TERM}\, t(v)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}] \\
&= (\llbracket (|_{u \in TERM}\, t(u)) \rrbracket \Gamma_L\ \Gamma_T) && [\text{provided } u \notin \phi t] \\
&= (\llbracket (\textbf{con } u \bullet t[v \setminus u]) \rrbracket \Gamma_L\ \Gamma_T) && [\text{Definition of } \textbf{con}]
\end{aligned}
$$

□

D.6 Lemmas

D.6.1 Lemma 3

We prove this Lemma by structural induction, using definition 3 above.

Base cases :

skip =!**skip** [Law 8]

fail =!**fail** [Law 9]

law l =!(**law** l) [Law 10]

Inductive step : We assume that t_i is sequential, this means t_i =!t_i

We must consider the following cases

a) !t

$= !(t \mid \textbf{fail})$ [Law 2]

$= !(!t \mid \textbf{fail})$ [Law 14]

$= !(!t)$ [Law 2]

b) **fails** t =! **fails** t [Law 22]

c) **succs** t =! **succs** t [Law 43]

d) $t_1; \ t_2$

$= !t_1; \ !t_2$ [Inductive Hypothesis]

$= !(!t_1; \ !t_2)$ [Law 12]

$= !(t_1; \ t_2)$ [Inductive Hypothesis]

In the following proofs we use the notation $list_i$ to denote the ith-element of the list *list*.

e) $\boxed{\textbf{if}} \ tactics \ \boxed{\textbf{fi}} \ (\textbf{if} \quad GC \ \textbf{fi}, proofs)$

$= \Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$

$\qquad\qquad (\Pi(apply \ (([\![\_]\!]\Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (extractP \ GC)))))$

$\qquad\qquad\qquad\qquad\qquad$ [Definition of $\boxed{\textbf{if}} \ \boxed{\textbf{fi}}$]

$= \Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$

$\qquad\qquad (\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$

$\qquad\qquad\qquad\qquad\qquad$ [Definition of *apply* and $*$]

$$= \Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(\llbracket!tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$$
$$\text{[Assumption]}$$

$$= \Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(head'(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$$
$$\text{[Definition of !]}$$

$$= \Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$head'(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$$
$$\text{[Lemma 18]}$$

$$= (if \ (proofs)) * (mkGC \ (extractG \ GC)$$
$$head'(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$$
$$\text{[Definition of } \Omega_{if}]$$

$$= (if \ (proofs)) * ((insertG \ (extractG \ GC))*$$
$$head'(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$$
$$\text{[Definition of } mkGC]$$

$$= (if \ (proofs)) * (head'((insertG \ (extractG \ GC))*$$
$$(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$$
$$\text{[Lemma 17]}$$

$$= (if \ (proofs)) * (head'(mkGC(extractG \ GC)$$
$$(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$$
$$\text{[Definition of } mkGC]$$

$$= head'((if \ (proofs)) * (mkGC \ (extractG \ GC)$$
$$(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$$
$$\text{[Lemma 17]}$$

$$= head'(\Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$$
$$\text{[Definition of } \Omega_{if} \text{ and } mkGC]$$

$$= head'(\Omega_{if} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(apply \ ((\llbracket\_\rrbracket\Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (extractP \ GC)))))))$$
$$\text{[Definition of } apply \text{ and } *]$$

$$= \ !\boxed{\textbf{if}} \ tactics \ \boxed{\textbf{fi}} \ (\textbf{if} \ \ GC \ \textbf{fi}, proofs)$$
$$\text{[Definition of } \boxed{\textbf{if}} \ \boxed{\textbf{fi}} \text{ and } head']$$

f) $\boxed{\textbf{do}} \ tactics \ \boxed{\textbf{od}} \ (\textbf{do} \ \ GC \ \textbf{od}, proofs)$
$$= \Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(apply \ ((\llbracket\_\rrbracket\Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (extractP \ GC)))))$$
$$\text{[Definition of } \boxed{\textbf{do}} \ \boxed{\textbf{od}}]$$

$$= \Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$$
$$(\Pi(\llbracket tactics_i\rrbracket\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$$
$$\text{[Definition of } apply \text{ and } *]$$

$= \Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad (\Pi([\![!tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Assumption]

$= \Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad (\Pi(head'([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of !]

$= \Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad head'(\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Lemma 18]

$= (do \ (proofs)) * (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad head'(\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of Ω_{do}]

$= (do \ (proofs)) * ((insertG \ (extractG \ GC))*$
$\qquad\qquad\quad head'(\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $mkGC$]

$= (do \ (proofs)) * (head'((insertG \ (extractG \ GC))*$
$\qquad\qquad\quad (\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Lemma 17]

$= (do \ (proofs)) * (head'(mkGC(extractG \ GC)$
$\qquad\qquad\quad (\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $mkGC$]

$= head'((do \ (proofs)) * (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad (\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Lemma 17]

$= head'(\Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad (\Pi([\![tactics_i]\!]\Gamma_L \ \Gamma_T) \ (\Phi_r * extractP \ GC)_i)))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of Ω_{do} and mkGC]

$= head'(\Omega_{do} \ (proofs) \ (mkGC \ (extractG \ GC)$
$\qquad\qquad\quad (\Pi(apply \ (([\![\_]\!]\Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (extractP \ GC))))))))$
$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $apply$ and $*$]

$= ! \boxed{\textbf{do}} \ tactics \ \boxed{\textbf{od}} \ (\textbf{do} \ \ GC \ \textbf{od}, proofs)$

$\qquad\qquad\qquad\qquad\qquad\qquad$ [Definition of $\boxed{\textbf{do}} \ \boxed{\textbf{od}}$ and $head'$]

g) $\boxed{\textbf{var}} \ t_1 \,]\!]\!] \ (p, pobs)$

$= (var \ d) * ([\![t]\!]\Gamma_L\Gamma_T \ (p, pobs))$ $\qquad\qquad$ [Definition of $\boxed{\textbf{var}} \ ]\!]\!]$]

$= (var \ d) * ([\![!t]\!]\Gamma_L\Gamma_T \ (p, pobs))$ $\qquad\qquad$ [Assumption]

$= (var \ d) * (head'([\![t]\!]\Gamma_L\Gamma_T \ (p, pobs)))$ $\qquad$ [Definition of !]

$= head'((var \ d) * ([\![t]\!]\Gamma_L\Gamma_T \ (p, pobs)))$ $\qquad$ [Lemma 17]

$= ! \boxed{\textbf{var}} \ t_1 \,]\!]\!] \ (p, pobs)$ $\qquad\qquad\qquad\qquad$ [Definition of ! and $\boxed{\textbf{var}} \ ]\!]\!]$]

h) $\boxed{\textbf{con}} \ t_1 \,]\!]\!] \ (p, pobs)$

$= (cons\ d) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Definition of $\llbracket \mathbf{con} \rrbracket$]

$= (cons\ d) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Assumption]

$= (cons\ d) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Definition of !]

$= head'((cons\ d) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Lemma 17]

$= \llbracket !\mathbf{con}\ t_1 \rrbracket\ (p, pobs)$ \hfill [Definition of ! and $\llbracket \mathbf{con} \rrbracket$]

i) $\llbracket \mathbf{pmain}\ t_1 \rrbracket\ (p, pobs)$

$= (procm\ n\ p) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Definition of $\llbracket \mathbf{pmain} \rrbracket$]

$= (procm\ n\ p) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Assumption]

$= (procm\ n\ p) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Definition of !]

$= head'((procm\ n\ p) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Lemma 17]

$= \llbracket !\mathbf{pmain}\ t_1 \rrbracket\ (p, pobs)$ \hfill [Definition of ! and $\llbracket \mathbf{pmain} \rrbracket$]

j) $\llbracket \mathbf{pmainvariant}\ t_1 \rrbracket\ (p, pobs)$

$= (variant\ n\ b\ v\ e) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Definition of $\llbracket \mathbf{pmainvariant} \rrbracket$]

$= (variant\ n\ b\ v\ e) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Assumption]

$= (variant\ n\ b\ v\ e) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Definition of !]

$= head'((variant\ n\ b\ v\ e) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Lemma 17]

$= \llbracket !\mathbf{pmainvariant}\ t_1 \rrbracket\ (p, pobs)$ \hfill [Definition of !]

\hfill [Definition of $\llbracket \mathbf{pmainvariant} \rrbracket$]

k) $\llbracket \mathbf{pbody}\ t_1 \rrbracket\ (p, pobs)$

$= (procb\ n\ p) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Definition of $\llbracket \mathbf{pbody} \rrbracket$]

$= (procb\ n\ p) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Assumption]

$= (procb\ n\ p) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Definition of !]

$= head'((procb\ n\ p) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Lemma 17]

$= \llbracket !\mathbf{pbody}\ t_1 \rrbracket\ (p, pobs)$ \hfill [Definition of ! and $\llbracket \mathbf{pbody} \rrbracket$]

l) $\llbracket \mathbf{pbodyvariant}\ t_1 \rrbracket\ (p, pobs)$

$= (variantb\ n\ b\ v\ e) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Definition of $\llbracket \mathbf{pbodyvariant} \rrbracket$]

$= (variantb\ n\ b\ v\ e) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ \hfill [Assumption]

$= (variantb\ n\ b\ v\ e) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Definition of !]

$= head'((variantb\ n\ b\ v\ e) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ \hfill [Lemma 17]

$= \llbracket !\mathbf{pbodyvariant}\ t_1 \rrbracket\ (p, pobs)$ \hfill [Definition of !]

\hfill [Definition of $\llbracket \mathbf{pbodyvariant} \rrbracket$]

m) $\llbracket \mathbf{pbodymain}\ t_b\ t_m \rrbracket\ (p, pobs)$

$= \llbracket \mathbf{pbody}\ t_b \rrbracket\ ;\ \llbracket \mathbf{pmain}\ t_m \rrbracket\ (p, pobs)$ \hfill [Definition of $\llbracket \mathbf{pbodymain} \rrbracket$]

$= \llbracket !\mathbf{pbody}\ t_b \rrbracket\ ;\ \llbracket !\mathbf{pmain}\ t_m \rrbracket\ (p, pobs)$ \hfill [Items (i) and (k)]

$= !(\llbracket !\mathbf{pbody}\ t_b \rrbracket\ ;\ \llbracket !\mathbf{pmain}\ t_m \rrbracket)\ (p, pobs)$ \hfill [Law 12]

$= !(\llbracket \mathbf{pbody}\ t_b \rrbracket\ ;\ \llbracket \mathbf{pmain}\ t_m \rrbracket)\,(p, pobs)$ [Items (i) and (k)]

$= !\llbracket \mathbf{pbodymain}\ t_b\ t_m \rrbracket\,(p, pobs)$ [Definition of $\llbracket \mathbf{pbodymain}\ \rrbracket$]

n) $\llbracket \mathbf{pmainvariantbody}\ t_1 \rrbracket\,(p, pobs)$

$= \llbracket \mathbf{pbodyvariant}\ t_b \rrbracket\ ;\ \llbracket \mathbf{pmainvariant}\ t_m \rrbracket\,(p, pobs)$

     [Definition of $\llbracket \mathbf{pmainvariantbody}\ \rrbracket$]

$= !\llbracket \mathbf{pbodyvariant}\ t_b \rrbracket\ ;\ !\llbracket \mathbf{pmainvariant}\ t_m \rrbracket\,(p, pobs)$

     [Items (j) and (l)]

$= !(!\llbracket \mathbf{pbodyvariant}\ t_b \rrbracket\ ;\ !\llbracket \mathbf{pmainvariant}\ t_m \rrbracket)\,(p, pobs)$

     [Law 12]

$= !(\llbracket \mathbf{pbodyvariant}\ t_b \rrbracket\ ;\ \llbracket \mathbf{pmainvariant}\ t_m \rrbracket)\,(p, pobs)$

     [Items (j) and (l)]

$= !\llbracket \mathbf{pmainvariantbody}\ t_1 \rrbracket\,(p, pobs)$ [Definition of $\llbracket \mathbf{pmainvariant}\ \rrbracket$]

o) $\llbracket \mathbf{val}\ t_1 \rrbracket\,(p, pobs)$

$= (val\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Definition of $\llbracket \mathbf{val}\ \rrbracket$]

$= (val\ v\ t\ a) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Assumption]

$= (val\ v\ t\ a) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Definition of !]

$= head'((val\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Lemma 17]

$= !\llbracket \mathbf{val}\ t_1 \rrbracket\,(p, pobs)$ [Definition of ! and $\llbracket \mathbf{val}\ \rrbracket$]

p) $\llbracket \mathbf{res}\ t_1 \rrbracket\,(p, pobs)$

$= (res\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Definition of $\llbracket \mathbf{res}\ \rrbracket$]

$= (res\ v\ t\ a) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Assumption]

$= (res\ v\ t\ a) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Definition of !]

$= head'((res\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Lemma 17]

$= !\llbracket \mathbf{res}\ t_1 \rrbracket\,(p, pobs)$ [Definition of ! and $\llbracket \mathbf{res}\ \rrbracket$]

q) $\llbracket \mathbf{val\text{-}res}\ t_1 \rrbracket\,(p, pobs)$

$= (valres\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Definition of $\llbracket \mathbf{val\text{-}res}\ \rrbracket$]

$= (valres\ v\ t\ a) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, pobs))$ [Assumption]

$= (valres\ v\ t\ a) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Definition of !]

$= head'((valres\ v\ t\ a) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, pobs)))$ [Lemma 17]

$= !\llbracket \mathbf{val\text{-}res}\ t_1 \rrbracket\,(p, pobs)$ [Definition of ! and $\llbracket \mathbf{val\text{-}res}\ \rrbracket$]

r) $\llbracket \mathbf{parcommand}\ t_1 \rrbracket\,(p, post)$

$= (parcommand\ d) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, post))$ [Definition of $\llbracket \mathbf{parcommand}\ \rrbracket$]

$= (parcommand\ d) * (\llbracket !t \rrbracket \Gamma_L \Gamma_T\ (p, post))$ [Assumption]

$= (parcommand\ d) * (head'(\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, post)))$ [Definition of !]

$= head'((valres\ d) * (\llbracket t \rrbracket \Gamma_L \Gamma_T\ (p, post)))$ [Lemma 17]

$= \,!\boxed{\textbf{parcommand}}\, t_1 \,\boxed{]\!]}\, (p, post)$ [Definition of !]

  [Definition of $\boxed{\textbf{parcommand}}\,\boxed{]\!]}$]

□

D.6.2 Lemma 7

$$f* \bullet \,\bowtie\!/ = \,\bowtie\!/ \bullet f**$$

This Lemma can be proved by induction as follows:

Base Case : $lst = [\,]$
$f* \bullet \,\bowtie\!/[\,]$
$= f* \,\bowtie\!/[\,]$ [Functional Composition]
$= f*[\,]$ [Definition of $\bowtie\!/$]
$= [\,]$ [Definition of $*$]
$= \,\bowtie\!/[\,]$ [Definition of $\bowtie\!/$]
$= \,\bowtie\!/ f**[\,]$ [Definition of $*$]

Base Case : $lst = \bot$
$f* \bullet \,\bowtie\!/[\,]$
$= f* \bot$ [Definition of $\bowtie\!/$]
$= \bot$ [Definition of $*$]
$= \,\bowtie\!/ \bot$ [Definition of $\bowtie\!/$]
$= \,\bowtie\!/ f** \bot$ [Definition of $*$]

Base Case : $lst = [[x]]$
$f* \bullet \,\bowtie\!/[[x]]$
$= f*[x]$ [Definition of $\bowtie\!/$]
$= [f\ x]$ [Definition of $*$]
$= \,\bowtie\!/[[f\ x]]$ [Definition of $\bowtie\!/$]
$= \,\bowtie\!/ f** [[x]]t$ [Definition of $*$]

Inductive step:
$fs : \mathrm{finseq}(\mathrm{pfiseq}\ X)$
$fs : \mathrm{pfiseq}(\mathrm{pfiseq}\ X)$

We assume that
$f* \bullet \,\bowtie\!/ fs = \,\bowtie\!/ \bullet f** fs.$
$f* \bullet \,\bowtie\!/ is = \,\bowtie\!/ \bullet f** is.$

We must prove that $f* \bullet \,\bowtie\!/ fs \,\bowtie\, is = \,\bowtie\!/ \bullet f** fs \,\bowtie\, is.$

167

$f * \bullet \; ⋈\!/ \, fs \; ⋈ \; is$
$= (f * \bullet \; ⋈\!/ \, fs) \; ⋈ \; (f * \bullet \; ⋈\!/ \, is)$ [Lemma D.6.21]
$= (⋈\!/ \bullet f * * \, fs) \; ⋈ \; (⋈\!/ \bullet f * * \, is)$ [Assumption]
$= ⋈\!/((f * * \, fs) \; ⋈ \; (f * * \, is))$ [Lemma D.6.5]
$= ⋈\!/ \bullet f * *(fs \; ⋈ \; is)$ [Lemma D.6.12]

$\square$

D.6.3 Lemma 8

$$⋈\!/ \bullet ⋈\!/ = ⋈\!/ \bullet ⋈\!/ *$$

This lemma is also proved by induction as seen below:

$⋈\!/ \bullet ⋈\!/ = ⋈\!/ \bullet ⋈\!/ *$

Base Case: lst = [ ]
$⋈\!/ \bullet ⋈\!/[\,]$
$= ⋈\!/ \bullet ⋈\!/ * [\,]$ [Definition of $*$]

Base Case: lst = $\perp$
$⋈\!/ \bullet ⋈\!/[\,]$
$= \perp$ [Definition of $⋈\!/$]
$= ⋈\!/ \bullet ⋈\!/ * \perp$ [Definition of $⋈\!/,*$]

Base Case: lst = $[[[x]]]$
$⋈\!/ \bullet ⋈\!/[[[x]]]$
$= [x]$ [Definition of $*$]
$= ⋈\!/[[x]]$ [Definition of $*$]
$= ⋈\!/ \bullet ⋈\!/ * [[[x]]]$ [Definition of $*$]

Inductive step:
$fs :$ finseq(pfiseq X)
$fs :$ pfiseq(pfiseq X)

We assume that
$⋈\!/ \bullet ⋈\!/ \, fs = ⋈\!/ \bullet ⋈\!/ * \, fs$
$⋈\!/ \bullet ⋈\!/ \, is = ⋈\!/ \bullet ⋈\!/ * \, is$

We must prove that $⋈\!/ \bullet ⋈\!/ \, fs \; ⋈ \; is = ⋈\!/ \bullet ⋈\!/ * \, fs \; ⋈ \; is$

168

$$\bowtie/ \bullet \bowtie/ fs \bowtie is$$
$$= \bowtie/(\bowtie/ fs \bowtie \bowtie/ is) \qquad\qquad \text{[Lemma D.6.5]}$$
$$= (\bowtie/ \bullet \bowtie/ fs) \bowtie (\bowtie/ \bullet \bowtie/ is) \qquad\qquad \text{[Lemma D.6.5]}$$
$$= (\bowtie/ \bullet \bowtie/ * fs) \bowtie (\bowtie/ \bullet \bowtie/ * is) \qquad\qquad \text{[Assumption]}$$
$$= \bowtie/((\bowtie/ * fs) \bowtie (\bowtie/ * is)) \qquad\qquad \text{[Lemma D.6.5]}$$
$$= \bowtie/ \bullet \bowtie/ *(fs \bowtie is) \qquad\qquad \text{[Lemma D.6.12]}$$

$\square$

D.6.4 Lemma 9

$$head'\ l = l \ \ if \ l = [\,]\ or\ \#l = 1$$

See Lemma D.6.20.

D.6.5 Lemma 10

$$\bowtie/(a \bowtie b) = \bowtie/ a \bowtie \bowtie/ b$$

This lemma is proved by induction as follows:

$$\bowtie/(a \bowtie b) = \bowtie/ a \bowtie \bowtie/ b$$

Base Case: $a = [\,]$
$$\bowtie/(a \bowtie b)$$
$$= \bowtie/([\,] \bowtie b)$$
$$= \bowtie/ b \qquad\qquad \text{[Definition of } \bowtie]$$
$$= [\,] \bowtie \bowtie/ b \qquad\qquad \text{[Definition of } \bowtie]$$
$$= \bowtie/[\,] \bowtie \bowtie/ b \qquad\qquad \text{[Definition of } \bowtie/]$$

Base Case: $a = \perp$
$$\bowtie/(a \bowtie b)$$
$$= \perp \qquad\qquad \text{[Definition of } \bowtie, \bowtie/]$$
$$= \perp \bowtie \bowtie/ b \qquad\qquad \text{[Definition of } \bowtie]$$
$$= \bowtie/ \perp \bowtie \bowtie/ b \qquad\qquad \text{[Definition of } \bowtie/]$$

Base Case: $a = [e]$
$$\bowtie/(a \bowtie b)$$
$$= \bowtie/([e] \bowtie b)$$
$$= \bigsqcup_{\infty}\{c : [e] \bowtie b \bullet {}^{\wedge} c\} \qquad\qquad \text{[Definition of } \bowtie/]$$
$$= \bigsqcup_{\infty}\{c : \{x : [e];\ y : b \bullet x {}^{\wedge} y\} \bullet {}^{\wedge} c\} \qquad\qquad \text{[Definition of } \bowtie]$$
$$= \{c : \{x : [e];\ y : b \bullet x {}^{\wedge} y\} \bullet {}^{\wedge} c\} \qquad\qquad \text{[Definition of } \bigsqcup_{\infty}]$$

169

We need to make induction also on the list b.

Base Case: $b = [\,]$

$$= \{\, c : \{x : [e];\ y : [\,] \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\}$$
$$= \{\, c : \{x : [e] \bullet x\} \bullet \hat{}\ c\,\} \qquad\qquad \text{[Definition of } \hat{}\,]$$
$$= \{\, c : [e] \bullet \hat{}\ c\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : [\,] \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : b \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : {\bowtie\!}/\, c;\ y : {\bowtie\!}/\, b \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Definition of } {\bowtie\!}/\,]$$
$$= {\bowtie\!}/[e] \;\bowtie\; {\bowtie\!}/\, b \qquad\qquad \text{[Definition of } \bowtie\,]$$

Base Case: $b = \bot$

$$= \{\, c : \{x : [e];\ y : \bot \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\}$$
$$= \bot \qquad\qquad \text{[Definition of } \hat{}\,]$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \bot \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Definition of } \hat{}\,]$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : \bot \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : b \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : {\bowtie\!}/\, c;\ y : {\bowtie\!}/\, b \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Definition of } {\bowtie\!}/\,]$$
$$= {\bowtie\!}/[e] \;\bowtie\; {\bowtie\!}/\, b \qquad\qquad \text{[Definition of } \bowtie\,]$$

Base Case: $b = [v]$

$$= \{\, c : \{x : [e];\ y : [v] \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\}$$
$$= \{\, c : e \,\hat{}\, v \bullet \hat{}\ c\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : [v] \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : b \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Set theory]}$$
$$= \{\, x : {\bowtie\!}/\, c;\ y : {\bowtie\!}/\, b \bullet x \,\hat{}\, y\,\} \qquad\qquad \text{[Definition of } {\bowtie\!}/\,]$$
$$= {\bowtie\!}/[e] \;\bowtie\; {\bowtie\!}/\, b \qquad\qquad \text{[Definition of } \bowtie\,]$$

Inductive Step on b:

$hs : \text{finseq } X$

$is : \text{pfiseq } X$

We assume that
$$\{\, c : \{x : [e];\ y : hs \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\} = \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : hs \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\}$$
$$\{\, c : \{x : [e];\ y : is \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\} = \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : is \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\}$$

We must prove that
$$\{\, c : \{x : [e];\ y : hs \bowtie is \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\} = \{\, x : \{c : [e] \bullet \hat{}\,c\};\ y : \{c : hs \bowtie is \bullet \hat{}\,c\} \bullet x \,\hat{}\, y\,\}$$

$$\{\, c : \{x : [e];\ y : hs \bowtie is \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\}$$
$$= \{\, c : \{x : [e];\ y : hs \bullet x \,\hat{}\, y\} \cup \{x : [e];\ y : is \bullet x \,\hat{}\, y\} \bullet \hat{}\ c\,\} \qquad \text{[Set Comprehension]}$$

$$= \{\, c : \{x : [e];\ y : hs \bullet x \,\hat{}\, y\} \bullet \,\hat{}\, c\} \cup \{\, c : \{x : [e];\ y : hs \bullet x \,\hat{}\, y\} \bullet \,\hat{}\, c\}$$
$$\hspace{8cm} \text{[Set Comprehension]}$$
$$= \{x : \{c : [e] \bullet \,\hat{}\, c\};\ y : \{c : hs \bullet \,\hat{}\, c\} \bullet x \,\hat{}\, y\} \cup \{x : \{c : [e] \bullet \,\hat{}\, c\};\ y : \{c : is \bullet \,\hat{}\, c\} \bullet x \,\hat{}\, y\}$$
$$\hspace{9cm} \text{[Assumption]}$$
$$= \{x : \{c : [e] \bullet \,\hat{}\, c\};\ y : \{c : hs \bowtie is \bullet \,\hat{}\, c\} \bullet x \,\hat{}\, y\} \hspace{2cm} \text{[Set theory]}$$

Inductive Step on a:
$fs : \text{finseq } X$
$gs : \text{pfiseq } X$
We assume that
$$\{c : \{x : fs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\} = \{x : \{c : fs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\}$$
$$\{c : \{x : gs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\} = \{x : \{c : gs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\}$$

We must prove that
$$\{c : \{x : fs \bowtie gs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\} = \{x : \{c : fs \bowtie gs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\}$$

$$\{c : \{x : fs \bowtie gs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\}$$
$$= \{c : \{x : fs;\ y : b \bullet x \,\hat{}\, y\} \cup \{x : gs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\} \hspace{1cm} \text{[Set Comprehension]}$$
$$= \{c : \{x : fs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\, c\} \cup \{c : \{x : gs;\ y : b \bullet x \,\hat{}\, y\} \bullet \,\hat{}\ c\}$$
$$\hspace{9cm} \text{[Set Comprehension]}$$
$$= \{x : \{c : fs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\} \cup \{x : \{c : gs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\}$$
$$\hspace{9cm} \text{[Assumption]}$$
$$= \{x : \{c : fs \bowtie gs \bullet \,\hat{}\, c\};\ y : b \bullet x \,\hat{}\, y\} \hspace{2cm} \text{[Set Comprehension]}$$

$$\square$$

D.6.6   Lemma 11

$$head'(l \bowtie ((\bowtie/ \bullet f*)\ l)) = head'\ l$$

This lemma is also proved by induction as seen below:

$$head'(l \bowtie ((\bowtie/ \bullet f*)\ l)) = head'\ l$$

Base Case: $l = [\,]$
$$head'([\,] \bowtie ((\bowtie/ \bullet f*)\ [\,]))$$
$$= head'([\,] \bowtie (\bowtie/[\,])) \hspace{3cm} \text{[Definition of } *]$$
$$= head'([\,] \bowtie [\,]) \hspace{3.5cm} \text{[Definition of } \bowtie/]$$
$$= head'([\,]) \hspace{4.5cm} \text{[Definition of } \bowtie]$$

Base Case: $l = \bot$
$$head'(\bot \bowtie ((\bowtie/ \bullet f*)\ \bot))$$

$$= head'(\bot) \qquad\qquad\qquad\qquad\qquad\qquad\text{[Definition of } \bowtie]$$

Base Case: $l = [x]$
$head'([x] \bowtie ((\bowtie\!/ \bullet f*)\ [x]))$
$$= head'([x] \bowtie (\bowtie\!/[x])) \qquad\qquad\qquad\qquad \text{[Definition of } *]$$
$$= head'([x] \bowtie [x]) \qquad\qquad\qquad\qquad\qquad \text{[Definition of } \bowtie/]$$
$$= head'([x]) \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } \bowtie]$$

Inductive Step on a:
fs : finseq X
is : pfiseq X
Inductive step:
We assume that
$head'(fs \bowtie ((\bowtie\!/ \bullet f*)\ fs)) = head'\ fs$
$head'(is \bowtie ((\bowtie\!/ \bullet f*)\ is)) = head'\ is$

We must prove that
$head'((fs \bowtie is) \bowtie ((\bowtie\!/ \bullet f*)\ (fs \bowtie is))) = head'\ (fs \bowtie is)$

$head'((fs \bowtie is) \bowtie ((\bowtie\!/ \bullet f*)\ (fs \bowtie is)))$
$$= head'\ (fs \bowtie is) \qquad\qquad\qquad\qquad\qquad\quad \text{[Definition of } head']$$

$$\square$$

D.6.7 Lemma 12

$$head'(l_1 \bowtie l_2 \bowtie l_1) = head'(\bowtie\!/[l_1, l_2])$$

This lemma is proved by induction as follows:

$head'(l_1 \bowtie l_2 \bowtie l_1) = head'(\bowtie\!/[l_1, l_2])$

Base Case:
$l_1 = [\ ]$
$$head'([\ ] \bowtie l_2 \bowtie [\ ]) = head'(l_2) \qquad\qquad\qquad \text{[Definition of } \bowtie]$$
$$= head'(\bowtie\!/[[\ ], l_2]) \qquad\qquad\qquad\qquad\quad\ \text{[Definition of } \bowtie/]$$

$l_1 = \bot$
$$head'(\bot \bowtie l_2 \bowtie \bot) = head'(\bot) \qquad\qquad\qquad\quad \text{[Definition of } \bowtie]$$
$$= head'(\bowtie\!/[\bot, l_2]) \qquad\qquad\qquad\qquad\quad\ \text{[Definition of } \bowtie/]$$

Inductive step:

$fs : \text{finseq } X$
$is : \text{pfiseq } X$

We assume that
$head'(fs \bowtie l_2 \bowtie fs) = head'(\bowtie/[fs, l_2])$
$head'(is \bowtie l_2 \bowtie is) = head'(\bowtie/[is, l_2])$

We must prove that
$head'((fs \bowtie is) \bowtie l_2 \bowtie (fs \bowtie is)) = head'(\bowtie/[(fs \bowtie is), l_2])$

$head'((fs \bowtie is) \bowtie l_2 \bowtie (fs \bowtie is))$
$= head'(fs \bowtie is)$ $\hfill$ [Definition of $head'$]
$= head'((fs \bowtie is) \bowtie l_2)$ $\hfill$ [Definition of $\bowtie$]
$= head'(\bowtie/[(fs \bowtie is), l_2])$ $\hfill$ [Definition of $\bowtie/$]

$\hfill \square$

D.6.8 Lemma 13

$[a : as \mid a \leftarrow A, as \leftarrow B] \, and \, \forall \, l : B \bullet \#l = 1$
$= [< a, b >\mid a \leftarrow A, b \leftarrow B]$

This lemma is proved by induction as follows:

Base Case:
$B = [\,]$
$[a : as \mid a \leftarrow A, as \leftarrow B] = [a : as \mid a \leftarrow A, as \leftarrow>$
$= [\,]$ $\hfill$ [List Comprehension]
$= [< a, b >\mid a \leftarrow A, b \leftarrow]$ $\hfill$ [List Comprehension]
$= [< a, b >\mid a \leftarrow A, b \leftarrow B]$

Inductive step: We assume that
$[a : as \mid a \leftarrow A, as \leftarrow B] \wedge \forall \, l : B \bullet \#l = 1 \, =\, << a, b >\mid a \leftarrow A, b \leftarrow B >$

We must prove that
$[a : as \mid a \leftarrow A,$
$\qquad as \leftarrow \{< x >\} \cup B > \wedge \forall \, l : \{< x >\} \cup B$
$\qquad \bullet \#l = 1 \, =\, << a, b >\mid a \leftarrow A, b \leftarrow \{< x >\} \cup B]$

$[a : as \mid a \leftarrow A, as \leftarrow \{< x >\} \cup B]$
$= [a : as \mid a \leftarrow A, as \leftarrow \{< x >\}]$
$\quad \bowtie [a : as \mid a \leftarrow A, as \leftarrow B]$ $\hfill$ [Property of $\bowtie$]

$$= [< a, x >| \ a \leftarrow A] \overset{\propto}{} [a : as \mid a \leftarrow A, as \leftarrow B] \qquad \text{[List Comprehension]}$$
$$= [< a, b >| \ a \leftarrow A, b \leftarrow \{< x >\}] \overset{\propto}{} [< a, b >| \ a \leftarrow A, b \leftarrow B] \qquad \text{[List Comprehension]}$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[Assumption]}$$
$$= [< a, b >| \ a \leftarrow A, b \leftarrow \{< x >\} \cup B] \qquad\qquad\qquad \text{[Property of } \overset{\propto}{}]$$

$$\square$$

D.6.9 Lemma 14

$$\forall \, t : Tactic, \forall (p, pr) \ \in \ [\![t]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$$

This lemma is proved by induction as seen below:

Base Cases:
$t = \textbf{law} \ lname(args)$

$[\![\, \textbf{law} \ \ l \ (args)]\!]\Gamma_L \ \Gamma_T \ (p', pr')$

This is defined as

$$\begin{aligned}
&if \ \ lname \in \mathrm{dom}\,\Gamma_L \wedge args \in \mathrm{dom}\,\Gamma_L \ lname \wedge p' \in \mathrm{dom}\,\Gamma_L \ lname \ args \ then \\
&\quad let \ (newp, proofs) = \Gamma_L \ lname \ args \ p' \ \in \\
&\quad [(newp, pr' \cup nproofs)] \\
&else \\
&\quad [\,]
\end{aligned}$$

As $pr' \subseteq pr' \cup proofs$ we can conclude that
$\forall (p, pr) \ \in \ [\![\, \textbf{law} \ \ l \ (args)]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$t = \textbf{skip}$

$[\![\textbf{skip}]\!]\Gamma_L \ \Gamma_T \ (p', pr') = [(\text{p'},\text{pr'})]$
So $\forall (p, pr) \ \in \ [\![\textbf{skip}]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$t = \textbf{fail}$

$[\![\textbf{fail}]\!]\Gamma_L \ \Gamma_T \ (p', pr') = [\ ]$
So $\forall (p, pr) \ \in \ [\![\textbf{fail}]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$t = \textbf{abort}$

$[\![\textbf{abort}]\!]\Gamma_L \ \Gamma_T \ (p', pr') = \perp$
So $\forall (p, pr) \ \in \ [\![\textbf{abort}]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

Inductive step: We assume that
$\forall\, t_1, t_2 : \textit{Tactic}$ and
$\forall (p, pr)\ \in\ \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$ and
$\forall (p, pr)\ \in\ \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$
We must prove that

$$\forall (p, pr)\ \in\ \llbracket t_1;\ t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$$

$\llbracket t_1;\ t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') = {}^{\bowtie}\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')$ [Definition of ;]

let $\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') = list_A$
By Assumption we have that
$\forall (p'', pr'')\ \in\ list_A \bullet pr' \subseteq pr''$
Now, let ${}^{\bowtie}\!/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) * list_A = list_B$
We have by Assumption and transitivity $(pr' \subseteq pr'' \wedge pr'' \subseteq pr''' \Rightarrow pr' \subseteq pr''')$
that $\forall (p''', pr''')\ \in\ list_B \bullet pr' \subseteq pr'''$

We must also prove that

$$\forall (p, pr)\ \in\ \llbracket t_1\ \big|\ t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$$

$\llbracket t_1\ \big|\ t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')$
$= {}^{\bowtie}\!/[\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T]^{\circ}(p', pr')$ [Definition of $\big|$]
$= (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')) \overset{\bowtie}{} (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr'))$ [Definition of $\circ$ and ${}^{\bowtie}\!/$]
If, by assumption,
$\forall (p, pr)\ \in\ \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$ and
$\forall (p, pr)\ \in\ \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$
so we can say that
$\forall (p, pr)\ \in\ \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \overset{\bowtie}{} \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$

We must also prove that

$$\forall (p, pr)\ \in\ \llbracket !t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$$

$\llbracket !t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')$
$= head'(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr'))$ [Definition of !]
$= [head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')]$ [Definition of $head'$]
If, by assumption, we have that
$\forall (p, pr)\ \in\ \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr') \bullet pr' \subseteq pr$ and
we also can say that
$\forall (p, pr)\ \in\ [head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr')] \bullet pr' \subseteq pr$ and

We must also prove that

$$\forall (p, pr) \ \in \ [\![t_1 \boxed{;} t_2]\!] \Gamma_L \ \Gamma_T \ (p_1';\ p_2', pr') \bullet pr' \subseteq pr$$

This structural combinator is defined as below

$$[\![t_1 \boxed{;} t_2]\!] \Gamma_L \ \Gamma_T \ (p_1';\ p_2', proofs) =$$
$$\Omega_{sc} * (\Pi(< ([\![t_1]\!] \Gamma_L \ \Gamma_T)(p_1', proofs),$$
$$([\![t_2]\!] \Gamma_L \ \Gamma_T)(p_2', proofs) >))$$

We know that
$$\Pi < [\![t_1]\!] \Gamma_L \ \Gamma_T(p_1', proofs), [\![t_2]\!] \Gamma_L \ \Gamma_T(p_2', proofs) >$$
$$= [< (p_1, pr_1), (p_2, pr_2) >$$
$$\quad | \ (p_1, pr_1) \leftarrow [\![t_1]\!] \Gamma_L \ \Gamma_T(p_1', proofs),$$
$$\quad (p_2, pr_2) \leftarrow [\![t_1]\!] \Gamma_L \ \Gamma_T(p_2', proofs)] \qquad \text{[Definition of } \Pi]$$
$$= list_P$$
and that
$$\Omega_{sc} * \ list_P$$
$$= < (p_1;\ p_2, pr_1 \ \owedge \ pr_2)$$
$$\quad | \ (p_1, pr_1) \leftarrow [\![t_1]\!] \Gamma_L \ \Gamma_T(p_1', proofs),$$
$$\quad (p_2, pr_2) \leftarrow [\![t_1]\!] \Gamma_L \ \Gamma_T(p_2', proofs) > \qquad \text{[Definition of } \Omega_{sc} \text{ and } *]$$
$$= list_\Omega$$
By assumption we have that
$$\forall (p_1, pr_1) \ \in \ [\![t_1]\!] \Gamma_L \ \Gamma_T \ (p_1', pr') \bullet pr' \subseteq pr_1 \text{ and}$$
$$\forall (p_2, pr_2) \ \in \ [\![t_2]\!] \Gamma_L \ \Gamma_T \ (p_2', pr') \bullet pr' \subseteq pr_2$$
So we can say that $\forall (p_1;\ p_2, pr_1 \ \owedge \ pr_2) \ \in \ list_\Omega \bullet pr' \subseteq pr_1 \ \owedge \ pr_2$

We must also prove that
$$\forall (p, pr) \ \in \ [\![\boxed{\mathbf{if}} \ tactics \ \boxed{\mathbf{fi}}]\!] \Gamma_L \ \Gamma_T (\mathbf{if} \ \ GC \ \ \mathbf{fi}, pr') \bullet pr' \subseteq pr$$

$$= [\![\boxed{\mathbf{if}} \ tactics \ \boxed{\mathbf{fi}}]\!] \Gamma_L \ \Gamma_T (\mathbf{if} \ \ GC \ \ \mathbf{fi}, pr')$$
$$= \Omega_{if} \ (pr')$$
$$\quad (mkGC \ (brG \ \ GC)$$
$$\quad \quad (\Pi(apply \ (([\![\_]\!] \Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (brP \ \ GC)))))) \quad \text{[Definition of } \boxed{\mathbf{if}} \ \ \boxed{\mathbf{fi}}]$$
$$= (if \ pr')*$$
$$\quad (mkGC \ (brG \ \ GC)$$
$$\quad \quad (\Pi(apply \ (([\![\_]\!] \Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (brP \ \ GC)))))) \quad \text{[Definition of } \Omega_{if}]$$
$$= [(\mathbf{if} \ \ gc \ \mathbf{fi}, pr' \cup nps)$$
$$\quad | \ (gc, nps) \leftarrow mkGC \ (brG \ \ GC)$$
$$\quad \quad (\Pi(apply \ (([\![\_]\!] \Gamma_L \ \Gamma_T) * tactics) \ (\Phi_r * (brP \ \ GC))))]$$
$$\text{[Definition of } if]$$

176

We can see that
$$\forall (p, pr) \in [(\mathbf{if}\ gc\ \mathbf{fi}, pr' \cup nps)$$
$$\mid (gc, nps) \leftarrow mkGC\ (brG\ GC)$$
$$(\Pi(apply\ (([\![\_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ GC)))))]$$
$$\bullet\ pr' \subseteq pr' \cup nps$$

We must also prove that
$$\forall (p, pr) \in [\![\boxed{\mathbf{do}}\ tactics\ \boxed{\mathbf{od}}]\!]\Gamma_L\ \Gamma_T(\mathbf{do}\ \ GC\ \ \mathbf{od}, pr') \bullet\ pr' \subseteq pr$$

$$= [\![\boxed{\mathbf{do}}\ tactics\ \boxed{\mathbf{od}}]\!]\Gamma_L\ \Gamma_T(\mathbf{do}\ \ GC\ \ \mathbf{od}, pr')$$
$$= \Omega_{do}\ (pr')$$
$$(mkGC\ (brG\ GC)$$
$$(\Pi(apply\ (([\![\_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ GC)))))$$
$$\text{[Definition of } \boxed{\mathbf{do}}\ \ \boxed{\mathbf{od}}]$$
$$= (do\ pr') *$$
$$(mkGC\ (brG\ GC)$$
$$(\Pi(apply\ (([\![\_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ GC)))))$$
$$\text{[Definition of } \Omega_{do}]$$
$$= [(\mathbf{do}\ \ gc\ \mathbf{od}, pr' \cup nps)$$
$$\mid (gc, nps) \leftarrow mkGC\ (brG\ GC)$$
$$(\Pi(apply\ (([\![\_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ GC))))]$$
$$\text{[Definition of } do]$$

We can see that
$$\forall (p, pr) \in [(\mathbf{do}\ \ gc\ \mathbf{od}, pr' \cup nps)$$
$$\mid (gc, nps) \leftarrow mkGC\ (brG\ GC)$$
$$(\Pi(apply\ (([\![\_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ GC))))]$$
$$\bullet\ pr' \subseteq pr' \cup nps$$

We must also prove that
$$\forall (p, pr) \in [\![\boxed{\mathbf{var}}\ t_1\ \boxed{]\!]}]\!]\Gamma_L\ \Gamma_T\ ([\![\ \mathbf{var}\ \ d \bullet p']\!], pr') \bullet\ pr' \subseteq pr$$

$$[\![\boxed{\mathbf{var}}\ t_1\ \boxed{]\!]}]\!]\Gamma_L\ \Gamma_T\ ([\![\ \mathbf{var}\ \ d \bullet p']\!], pr')$$
$$= (var\ d) * ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr')) \qquad\qquad \text{[Definition of } \boxed{\mathbf{var}}\ \ \boxed{]\!]}]$$
$$= [([\![\ \mathbf{var}\ \ d \bullet p]\!], pr) \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr'))] \qquad \text{[Definition of } var]$$

So, we can say that
$$\forall (p, pr) \in [([\![\ \mathbf{var}\ \ d \bullet p]\!], pr)$$
$$\mid (p, pr) \leftarrow ([\![t]\!]\Gamma_L\ \Gamma_T\ (p', pr'))] \bullet\ pr' \subseteq pr \qquad\qquad \text{[Assumption]}$$

We must also prove that

$\forall (p, pr) \in [\![\mathbf{con} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{con} \ d \bullet p']\!], pr') \bullet pr' \subseteq pr$

$[\![\mathbf{con} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{con} \ d \bullet p']\!], pr')$
$= (cons \ d) * ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\mathbf{con}$ $[\![\ ]\!]$]
$= [([\![\mathbf{con} \ d \bullet p]\!], pr) \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))]$ [Definition of $cons$]

So, we can say that
$\forall (p, pr) \in [([\![\mathbf{con} \ d \bullet p]\!], pr)$
$\qquad\qquad \mid (p, pr) \leftarrow ([\![t]\!]\Gamma_L \ \Gamma_T \ (p', pr'))] \bullet pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \in [\![\mathbf{pmain} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \bullet p']\!], pr') \bullet pr' \subseteq pr$

$[\![\mathbf{pmain} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \bullet p']\!], pr')$
$= (procm \ pname \ body) * ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\mathbf{pmain}$ $[\![\ ]\!]$]
$= [([\![\mathbf{proc} \ pname = body \bullet p]\!], pr)$
$\quad \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))]$ [Definition of $procm$]

So, we can say that
$\forall (p, pr) \in [([\![\mathbf{proc} \ pname = body \bullet p]\!], pr)$
$\qquad\qquad \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))] \bullet pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \in [\![\mathbf{pmainvariant} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \ \mathbf{variant} \ v \mathbf{is} \ e \bullet p']\!], pr')$
$\qquad \bullet pr' \subseteq pr$

$[\![\mathbf{pmainvariant} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \ \mathbf{variant} \ v \mathbf{is} \ e \bullet p']\!], pr')$
$= (variant \ pname \ body \ v \ e) * ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\mathbf{pmainvariant}$ $[\![\ ]\!]$]
$= [([\![\mathbf{proc} \ pname = body \ \mathbf{variant} \ v \mathbf{is} \ e \bullet p]\!], pr)$
$\quad \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))]$ [Definition of $variant$]

So, we can say that
$\forall (p, pr) \in [([\![\mathbf{proc} \ pname = body \ \mathbf{variant} \ v \mathbf{is} \ e \bullet p]\!], pr)$
$\qquad\qquad \mid (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))]$
$\qquad\qquad \bullet pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \in [\![\mathbf{pbody} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \bullet p']\!], pr') \bullet pr' \subseteq pr$

$[\![\mathbf{pbody} \ t_1]\!]\Gamma_L \ \Gamma_T \ ([\![\mathbf{proc} \ pname = body \bullet p']\!], pr')$
$= (procb \ pname \ p') * ([\![t_1]\!]\Gamma_L \ \Gamma_T \ (body, pr'))$ [Definition of $\mathbf{pbody}$ $[\![\ ]\!]$]

178

$= [(\llbracket \mathbf{proc}\ \ pname = body' \bullet p' \rrbracket, pr)$
$\quad | \ (body', pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr'))]$ 　　　　　　　[Definition of $procb$]

So, we can say that
$\forall (p, pr) \ \in\ [(\llbracket \mathbf{proc}\ \ pname = body' \bullet p' \rrbracket, pr)$
$\qquad\qquad | \ (body', pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p', pr'))] \bullet pr' \subseteq pr$ 　[Assumption]

We must also prove that
$\forall (p, pr) \ \in\ \llbracket \boxed{\mathbf{pbodyvariant}}\ t_1 \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body\ \mathbf{variant}\ \ v \mathbf{is}\ e$
$\qquad \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\mathbf{pbodyvariant}}\ t_1 \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body\ \mathbf{variant}\ \ v \mathbf{is}\ e \bullet p' \rrbracket, pr')$
$= (variantb\ pname\ v\ e\ p') * (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (body, pr'))$ 　[Definition of $\boxed{\mathbf{pbodyvariant}}\ \boxed{}$]
$= [(\llbracket \mathbf{proc}\ \ pname = body'\ \mathbf{variant}\ \ v \mathbf{is}\ e \bullet p' \rrbracket, pr)$
$\quad | \ (body', pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (body, pr'))]$ 　[Definition of $variantb$]

So, we can say that
$\forall (p, pr) \ \in\ [(\llbracket \mathbf{proc}\ \ pname = body'\ \mathbf{variant}\ \ v \mathbf{is}\ e \bullet p' \rrbracket, pr)$
$\qquad\qquad | \ (body', pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (body, pr'))]$
$\qquad\qquad \bullet pr' \subseteq pr$ 　　　　　　　[Assumption]

We must also prove that
$\forall (p, pr) \ \in\ \llbracket \boxed{\mathbf{pbodymain}}\ t_b\ t_m \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\mathbf{pbodymain}}\ t_b\ t_m \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body \bullet p' \rrbracket, pr')$
$= \boxed{\mathbf{pbody}}\ t_b \rrbracket;\ \boxed{\mathbf{pmain}}\ t_m \rrbracket$ 　　　　　　[Definition of $\boxed{\mathbf{pbodymain}}\ \boxed{}$]

So, as we have already proved for sequential composition and for both structural combinators, we have proved already for this structural combinator.

We must also prove that
$\forall (p, pr) \ \in\ \llbracket \boxed{\mathbf{pmainvariantbody}}\ t_1 \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body\ \mathbf{variant}\ \ v \mathbf{is}\ e$
$\qquad \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\mathbf{pmainvariantbody}}\ t_1 \rrbracket \rrbracket \Gamma_L\ \Gamma_T\ (\llbracket \mathbf{proc}\ \ pname = body\ \mathbf{variant}\ \ v \mathbf{is}\ e \bullet p' \rrbracket, pr')$
$= \boxed{\mathbf{pbodyvariant}}\ t_b \rrbracket;\ \boxed{\mathbf{pmainvariant}}\ t_m \rrbracket$ 　[Definition of $\boxed{\mathbf{pmainvariantbody}}\ \boxed{}$]

So, as we have already proved for sequential composition and for both structural combinators, we have proved already for this structural combinator.

We must also prove that
$\forall (p, pr) \ \in\ \llbracket \boxed{\mathbf{val}}\ t_1 \rrbracket \Gamma_L\ \Gamma_T\ ((\mathbf{val}\ \ v : t \bullet p')(pars), pr') \bullet pr' \subseteq pr$

$[\![\boxed{\textbf{val}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{val}\ \ v:t\bullet p')(pars),pr')$

$=(val\ v\ t\ pars)*([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))$ [Definition of $\boxed{\textbf{val}}\ \boxed{]\!]}$]

$=[((\textbf{val}\ v:t\bullet p)(pars),pr)\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]$ [Definition of val]

So, we can say that

$\forall(p,pr)\ \in\ [((\textbf{val}\ v:t\bullet p),pr)$
$\qquad\qquad\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]\bullet pr'\subseteq pr$ [Assumption]

We must also prove that

$\forall(p,pr)\ \in\ [\![\boxed{\textbf{res}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{res}\ \ v:t\bullet p')(pars),pr')\bullet pr'\subseteq pr$

$[\![\boxed{\textbf{res}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{res}\ \ v:t\bullet p')(pars),pr')$

$=(res\ v\ t\ pars)*([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))$ [Definition of $\boxed{\textbf{res}}\ \boxed{]\!]}$]

$=[((\textbf{res}\ v:t\bullet p)(pars),pr)\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]$ [Definition of res]

So, we can say that

$\forall(p,pr)\ \in\ [((\textbf{res}\ v:t\bullet p)(pars),pr)$
$\qquad\qquad\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]\bullet pr'\subseteq pr$ [Assumption]

We must also prove that

$\forall(p,pr)\ \in\ [\![\boxed{\textbf{val-res}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{val-res}\ \ v:t\bullet p')(pars),pr')\bullet pr'\subseteq pr$

$[\![\boxed{\textbf{val-res}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{val-res}\ \ v:t\bullet p')(pars),pr')$

$=(valres\ v\ t\ pars)*([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))$ [Definition of $\boxed{\textbf{val-res}}\ \boxed{]\!]}$]

$=[((\textbf{val-res}\ v:t\bullet p)(pars),pr)$
$\quad\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]$ [Definition of $valres$]

So, we can say that

$\forall(p,pr)\ \in\ [((\textbf{val-res}\ v:t\bullet p)(pars),pr)$
$\qquad\qquad\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]\bullet pr'\subseteq pr$ [Assumption]

We must also prove that

$\forall(p,pr)\ \in\ [\![\boxed{\textbf{parcommand}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((pars\bullet p'),pr')\bullet pr'\subseteq pr$

$[\![\boxed{\textbf{parcommand}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((pars\bullet p'),pr')$

$=(parcommand\ pars)*([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))$ [Definition of $\boxed{\textbf{parcommand}}\ \boxed{]\!]}$]

$=[((pars\bullet p),pr)$
$\quad\mid(p,pr)\leftarrow([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p',pr'))]$ [Definition of $parcommand$]

So, we can say that

$$\forall (p, pr) \in [((pars \bullet p), pr)$$
$$| (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr'))] \bullet pr' \subseteq p \text{[Assumption]}$$

We must also prove that
$$\forall (p, pr) \in \llbracket \textbf{tactic} \ name(\{argument\}^*) \rrbracket \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$$

$$\llbracket \textbf{tactic} \ name(args) \rrbracket \Gamma_L \ \Gamma_T \ (p', pr') =$$
$$\quad if \ name \in \text{dom} \ \Gamma_T \wedge args \in \text{dom} \ \Gamma_T \ tname \ then$$
$$\quad\quad \Gamma_T \ tname \ args \ (p', pr')$$
$$\quad else$$
$$\quad\quad [\,]$$

We know that $\Gamma_T : name \rightarrow \{argument\}^* \rightarrow Tactic$.
So, let $\Gamma_T \ name \ arguments = t_1$
We have then
$= \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')$ We can say that
$$\forall (p, pr) \in \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr \qquad\qquad \text{[Assumption]}$$

We must also prove that
$$\forall (p, pr) \in \mu \ variable \bullet tactic \ (p', pr') \bullet pr' \subseteq pr$$

We know, by definition that this is defined as the least upper bound of a set of tactics, and the least upper bound of a set of tactics is a tactic. Let t_1 be this tactic. So, by assumption, we finish this proof.

We must also prove that
$$\forall (p, pr) \in \llbracket \textbf{applies to} \quad program \ \textbf{do} \ t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$$

$\llbracket \textbf{applies to} \quad program \ \textbf{do} \ t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')$
$= \llbracket \textbf{con} \ program \bullet \ equals \ (program, pr') t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')$
$$\text{[Definition of \textbf{applies to} \_ \textbf{do} \_]}$$
$= \llbracket |_{v \in TERM}(equals \ (program, pr'); \ t_2)(v) \rrbracket \ \Gamma_L \ \Gamma_T \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')$
$$\text{[Definition os \textbf{con}]}$$
We can say that
$$\forall (p, pr) \in \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$$
$$\text{[Assumption]}$$

$\square$

181

D.6.10 Lemma 15a

$$\Pi < lst_a, lst_b > \bowtie \Pi < lst_a, lst_c > = \Pi < lst_a, lst_b \bowtie lst_c >$$

$\Pi < lst_a, lst_b > \bowtie \Pi < lst_a, lst_c >$

$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_b >$

$\quad \bowtie < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_c >$ [Definition of Π]

$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow (e2l * lst_b) \bowtie (e2l * lst_c) >$ [Property of $\bowtie$]

$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * (lst_b \bowtie lst_c) >$ [Property of $*$]

$= \Pi < lst_a, lst_b \bowtie lst_c >$ [Definition of Π]

$\square$

D.6.11 Lemma 15b

$$\Pi < lst_a, lst_c > \bowtie \Pi < lst_b, lst_c > = \Pi < lst_a \bowtie lst_b, lst_c >$$

$\Pi < lst_a, lst_c > \bowtie \Pi < lst_b, lst_c >$

$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_c >$

$\quad \bowtie < (a : as) \mid a \leftarrow lst_b, as \leftarrow e2l * lst_c >$ [Definition of Π]

$= < (a : as) \mid a \leftarrow lst_a \bowtie lst_b, as \leftarrow e2l * lst_c >$ [Property of $\bowtie$]

$= \Pi < lst_a \bowtie lst_b, lst_c >$ [Definition of Π]

$\square$

D.6.12 Lemma 16

$$f * a \bowtie f * b = f * (a \bowtie b)$$

This lemma is proved by induction as follows:

Base Case: $a = [\,]$

$f * [\,] \bowtie f * b$

$= f * b$ [Definition of $*, \bowtie$]

$= f * ([\,] \bowtie b)$ [Definition of $\bowtie$]

Base Case: $a = \bot$

$f * \bot \bowtie f * b$

$= \bot \bowtie f * b$ [Definition of $*$]

$= \bot$ [Definition of $\bowtie$]

$= f * (\bot)$ [Definition of $*$]

$= f * (\bot \bowtie b)$ [Definition of $\bowtie$]

Base Case: $a = [x]$

$f * [x] \bowtie f * b$

$= (f\ x) \bowtie f * b$ [Definition of $*$]

$= f * ([x] \bowtie b)$ [Definition of $*$]

Inductive step:

$fs : \text{finseq }\ X$

$is : \text{pfiseq }\ X$

We assume that

$f * fs \bowtie f * b = f * (a \bowtie b)$

$f * is \bowtie f * b = f * (a \bowtie b)$

We must prove that

$f * (fs \bowtie is) \bowtie f * b = f * ((fs \bowtie is) \bowtie b)$

$f * (fs \bowtie is) \bowtie f * b = \{x : f * (fs \bowtie is);\ y : f * b \bullet x \,{}^\wedge y\}$ [Definition of $\bowtie$]

$= \{x : f * \{u : fs;\ v : is \bullet u \,{}^\wedge v\};\ y : f * b \bullet x \,{}^\wedge y\}$ [Definition of $\bowtie$]

$= \{x : f * fs;\ y : f * b \bullet x \,{}^\wedge y\} \cup \{x : f * is;\ y : f * b \bullet x \,{}^\wedge y\}$ [Set Comprehension]

$= f * \{x : fs;\ y : b \bullet x \,{}^\wedge y\} \cup f * \{x : is;\ y : b \bullet x \,{}^\wedge y\}$ [Assumption]

$= f * \{x : \{u : fs;\ v : is \bullet u \,{}^\wedge v\};\ y : b \bullet x \,{}^\wedge y\}$ [Set Comprehension]

$= f * \{x : fs \bowtie is;\ y : b \bullet x \,{}^\wedge y\}$ [Definition of $\bowtie$]

$= f * ((fs \bowtie is) \bowtie b)$ [Definition of $\bowtie$]

$\square$

D.6.13   Lemma 17

$head'(f * lst) = f * (head'\ lst)$

This lemma is proved by induction as seen below:

$head'(f * lst) = f * (head'\ lst)$

Base Case: $lst = [\ ]$

$head'(f * [\ ])$

$= head'([\ ])$ [Definition of $*$]

$= [\ ]$ [Definition of $head'$]

$= f * [\ ]$ [Definition of $*$]

$= f * (head'([\ ]))$ [Definition of $head'$]

Base Case: $lst = \bot$

$head'(f * \bot)$

$= head'(\bot)$ [Definition of $*$]

$= \bot$ [Definition of $head'$]

$= f * \perp$ [Definition of $*$]

$= f * (head'(\perp))$ [Definition of $head'$]

Base Case: $lst = [x]$

$head'(f * [x])$

$= head'([f\ x])$ [Definition of $*$]

$= [f\ x]$ [Definition of $head'$]

$= f * [x]$ [Definition of $*$]

$= f * (head'([x]))$ [Definition of $head'$]

Inductive step:

fs : finseq X

is : pfiseq X

We assume that

$head'(f * fs) = f * (head'\ fs)$

$head'(f * is) = f * (head'\ is)$

We must prove that

$head'(f * (fs \mathbin{\reflectbox{$\propto$}} is)) = f * (head'\ (fs \mathbin{\reflectbox{$\propto$}} is))$

$head'(f * (fs \mathbin{\reflectbox{$\propto$}} is))$

$= head'(f * fs \mathbin{\reflectbox{$\propto$}} f * is)$ [Lemma D.6.6]

$= head'(f * fs)$ [Property of $head'$]

$= [head(f * fs)]$ [Definition of $head'$]

$= [f\ head(fs)]$ [Property of $*$,$head$]

$= f * [head(fs)]$ [Definition of $*$]

$= f * head'(fs)$ [Definition of $head'$]

$= f * (head'\ (fs \mathbin{\reflectbox{$\propto$}} is))$ [Property of $head'$]

$\square$

D.6.14   Lemma 18

$$head'(\Pi(llst)) = \Pi(head' \circ llst)$$

This lemma is also proved by induction as follows:

Base Case:

$llst = <>$

$head'(\Pi(llst))$

$= head'(\Pi(<>))$
$= head'([\,])$ [Definition of Π]
$= [\,]$ [Definition of $head'$]
$= \Pi(<>)$ [Definition of Π]
$= \Pi(head' \circ <>)$ [Definition of $*$]

Inductive step: We assume that $head'(\Pi(ls)) = \Pi(head' * ls)$
We must prove that $head'(\Pi(l : ls)) = \Pi(head' * (l : ls))$

$head'(\Pi(l : ls)) = head'([a : as \mid a \leftarrow l, as \leftarrow \Pi ls])$ [Definition of Π]
$= [a : as \mid a \leftarrow head'\ l, as \leftarrow \Pi\ head' \circ\ ls])$ [Property of List Comprehension]
$= \Pi(head'\ l : head' \circ\ ls)$ [Definition of Π]
$= \Pi(head' \circ\ l : ls)$ [Definition of $\circ$]

$\square$

D.6.15   Lemma 19

$$head'(head'(lst)) = head'(lst)$$

Case: $lst = [\,]$
$head'(head'\ [\,])$
$= head'([\,])$ [Definition of $head'$]

Case: $lst = \bot$
$head'(head'\ \bot)$
$= head'(\bot)$ [Definition of $head'$]

Case: $lst = [x] \mathbin{⧏⧐} xs$
$head'(head'\ ([x] \mathbin{⧏⧐} xs))$
$= head'([x])$ [Definition of $head'$]

$\square$

D.6.16   Lemma 20

$$\otimes\!/ \bullet t * \; lst \neq [\,] \Rightarrow \otimes\!/ \bullet (\textbf{succs}\ t) * \; lst \neq [\,]$$

$\otimes\!/ \bullet t * \; lst \neq [\,]$
$\Rightarrow \exists\, r : RMCell \bullet \exists\, i : \mathbb{N} \bullet lst[i] = r \wedge t\ r \neq [\,]$
$\Rightarrow \exists\, p : \text{pfiseq}(RMCell) \bullet \exists\, r : RMCell \bullet \exists\, i : \mathbb{N} \bullet snd(p)[i] = r \wedge t\ r \neq [\,]$
$\Rightarrow \textbf{succs}\ t\ r \neq [\,]$
$\Rightarrow \otimes\!/ \bullet (\textbf{succs}\ t) * \; lst \neq [\,]$

□

D.6.17   Lemma 21

$$\forall f, g : X \to \text{pfiseq}\ X$$
$$\otimes\!/[f, g]^\circ\ r = f\ r \otimes g\ r$$

$\otimes\!/[f, g]^\circ\ r$
$= \otimes\!/[f\ r, g\ r]$ \hfill [Definition of $^\circ$]
$= \bigsqcup_\infty \{ c : [f\ r, g\ r] \bullet {}^\wedge c \}$ \hfill [Definition of $\otimes\!/$]
$= \bigsqcup_\infty \{ {}^\wedge(f, < f\ r, g\ r >) \}$ \hfill [Set theory]
$= \bigsqcup_\infty \{ ({}^\wedge(f, < f\ r >)) \otimes ({}^\wedge(f, < g\ r >)) \}$ \hfill [Definition of $^\wedge$]
$= \bigsqcup_\infty \{ f\ r \otimes g\ r \}$ \hfill [Definition of $^\wedge$]
$= \bigcup \{ f\ r \otimes g\ r \}$ \hfill [Definition of $\bigsqcup_\infty$]
$= f\ r \otimes g\ r$ \hfill [Definition of $\bigcup$]

□

D.6.18   Lemma 22

$$\forall\, a, b, c : \text{pfseq}\ X \bullet a\, {}^\wedge (b\, {}^\wedge c) = (a\, {}^\wedge b)\, {}^\wedge c$$

$a\, {}^\wedge (b\, {}^\wedge c)$

Case $a = (f, x), b = (f, y), c = (f, z)$
$= (f, x)\, {}^\wedge ((f, y)\, {}^\wedge (f, z))$
$= (f, x)\, {}^\wedge (f, y\ cat\ z)$ \hfill [Definition of $^\wedge$]
$= (f, x\ cat\ (y\ cat\ z))$ \hfill [Definition of $^\wedge$]
$= (f, (x\ cat\ y)\ cat\ z)$ \hfill [Property of $\otimes$]
$= (f, x \otimes y)\, {}^\wedge (f, z)$ \hfill [Definition of $^\wedge$]

186

$$= ((f, x) \wedge (f, y)) \wedge (f, z) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

Case $a = (f, x)$, $b = (f, y)$, $c = (p, z)$
$$= (f, x) \wedge ((f, y) \wedge (p, z))$$
$$= (f, x) \wedge (p, y \ cat \ z) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= (p, x \ cat \ (y \ cat \ z)) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= (p, (x \ cat \ y) \ cat \ z) \qquad\qquad\qquad \text{[Property of } \bowtie \text{]}$$
$$= (f, x \bowtie y) \wedge (p, z) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

$$= ((f, x) \wedge (f, y)) \wedge (p, z) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

Case $a = (f, x)$, $b = (p, y)$, $c = (x : \{f, p\}, z)$
$$= (f, x) \wedge ((p, y) \wedge c)$$
$$= (f, x) \wedge (p, y) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= (p, x \ cat \ y) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= (p, x \ cat \ y) \ catpf \ c \qquad\qquad\qquad \text{[Property of } \bowtie \text{]}$$
$$= (f, x \bowtie y) \wedge (p, z) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

$$= ((f, x) \wedge (p, y)) \wedge c \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

Case $a = (p, x)$, $b = (x : \{f, p\}, y)$, $c = (y : \{f, p\}, z)$
$$= (p, x) \wedge (b \wedge c)$$
$$= (p, x) \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= (p, x) \wedge b \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= ((p, x) \wedge b) \wedge c \qquad\qquad\qquad \text{[Definition of } \wedge \text{]}$$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$

D.6.19 Lemma 23

$$\forall \ e_1, e_2 : \text{pfiseq} \ X \bullet \bowtie/ \bullet f* \bullet \bowtie/[e_1, e_2] = (\bowtie/ \bullet f * \ e_1) \bowtie (\bowtie/ \bullet f * \ e_2)$$

$$\bowtie/ \bullet f* \bullet \bowtie/[e_1, e_2]$$
$$= \bowtie/ \bullet f * \bigsqcup_{\infty} \{c : [e_1, e_2] \bullet {}^{\wedge}c\} \qquad\qquad \text{[Definition of } \bowtie/ \text{]}$$
$$= \bowtie/ \bullet f * \bigsqcup_{\infty} \{{}^{\wedge}(f, < e_1, e_2 >)\} \qquad\qquad \text{[Set Theory]}$$
$$= \bowtie/ \bullet f * \bigsqcup_{\infty} \{e_1 \bowtie e_2\} \qquad\qquad \text{[Definition of } \wedge \text{]}$$
$$= \bowtie/ \bullet f * (e_1 \bowtie e_2)\} \qquad\qquad \text{[Definition of } \bigsqcup_{\infty}, \bigcup]$$
$$= \bowtie/\{x : e_1 \bowtie e_2 \bullet pfmap \ f \ x\} \qquad\qquad \text{[Definition of } * \text{]}$$

$= {}^{\bowtie}\!/\{x : \{a : e_1;\ b : e_2 \bullet a \,{}^{\wedge}\, b\} \bullet \mathit{pfmap}\ f\ x\}$ [Definition of ${}^{\bowtie}$]

$= {}^{\bowtie}\!/\{x : \{a : e_1 \bullet \mathit{pfmap}\ f\ a\};\ y : \{b : e_2 \bullet \mathit{pfmap}\ f\ b\} \bullet x \,{}^{\bowtie}\, y\}$

                                                       [Set Comprehension]

$= \{x : {}^{\bowtie}\!/\{a : e_1 \bullet \mathit{pfmap}\ f\ a\};\ y : {}^{\bowtie}\!/\{b : e_2 \bullet \mathit{pfmap}\ f\ b\} \bullet x \,{}^{\wedge}\, y\}$

                                                       [Property of ${}^{\bowtie}\!/, {}^{\bowtie}, {}^{\wedge}$]

$= \{x : {}^{\bowtie}\!/ \bullet f * e_1;\ y : {}^{\bowtie}\!/ \bullet f * e_2 \bullet x \,{}^{\wedge}\, y\}$ [Definition of $*$]

$= ({}^{\bowtie}\!/ \bullet f * e_1) \,{}^{\bowtie}\, ({}^{\bowtie}\!/ \bullet f * e_2)$ [Definition of ${}^{\bowtie}$]

                                                             □

D.6.20   Lemma 24

$\mathit{take}\ 1\ l = l\ \mathit{if}\ l = [\,]\ \mathit{or}\ (\mathit{size}\ l = 1 \wedge \forall\, x \in L \bullet \mathit{size}\ (\mathit{snd}\ x) = 1)$

Case $l = [\,]$

$\mathit{take}\ 1\ [\,] = [\,]$ [Definition of take]

Case $l = [e]$

$\mathit{take}\ 1\ [e] = [e]$ [Definition of take]

                                                             □

D.6.21   Lemma 25

$(f * \bullet \,{}^{\bowtie}\!/\, a) \,{}^{\bowtie}\, (f * \bullet \,{}^{\bowtie}\!/\, b) = f * \bullet \,{}^{\bowtie}\!/(a \,{}^{\bowtie}\, b)$

$(f * \bullet \,{}^{\bowtie}\!/\, a) \,{}^{\bowtie}\, (f * \bullet \,{}^{\bowtie}\!/\, b)$

$= ({}^{\bowtie}\!/ \bullet f * * a) \,{}^{\bowtie}\, ({}^{\bowtie}\!/ \bullet f * * b)$ [Lemma D.6.2]

$= {}^{\bowtie}\!/(f * * a \,{}^{\bowtie}\, f * * b)$ [Lemma D.6.5]

$= {}^{\bowtie}\!/ \bullet f * *(a \,{}^{\bowtie}\, b)$ [Lemma D.6.12]

$= f * \bullet \,{}^{\bowtie}\!/(a \,{}^{\bowtie}\, b)$ [Lemma D.6.2]

                                                             □

D.6.22   Lemma 26

$\mathit{head}(x \,{}^{\bowtie}\, y) = \mathit{head}([\mathit{head}\ x] \,{}^{\bowtie}\, y), x \neq [\,]$

This lemma is proved by induction as follows

Base Case: $x = \bot$
$head(\bot \bowtie y)$
$= head([head\ \bot] \bowtie y)$ [Definitiof of $head$]

Base Case: $x = [e]$
$head([e] \bowtie y)$
$= head([e])$ [Property of $head$]
$= head([head\ [e]])$ [Definition of $head$]
$= head([head\ [e]] \bowtie y)$ [Property of $head$]

Inductive step:
$fs : \text{finseq}\ X$
$is : \text{pfiseq}\ X$

We assume that
$head(fs \bowtie y) = head([head\ fs] \bowtie y), fs \neq [\ ]$
$head(is \bowtie y) = head([head\ is] \bowtie y), is \neq [\ ]$

We must prove that
$head(fs \bowtie is \bowtie y) = head([head\ fs \bowtie is] \bowtie y), fs \bowtie is \neq [\ ]$

$head(fs \bowtie is \bowtie y) = head(fs)$ [Property of $head$]
$= head([head\ fs])$ [Lemma D.6.15]
$= head([head\ fs \bowtie is])$ [Property of $head$]
$= head([head\ fs \bowtie is] \bowtie y)$ [Property of $head$]

$\square$

D.6.23   Lemma 27

$$head(x \bowtie y) = head(x \bowtie [heady]), y \neq [\ ]$$

This lemma is proved by induction as follows

Base Case: $y = \bot$
$head(x \bowtie \bot)$
$= head(x \bowtie [head\ \bot])$ [Definitiof of $head$]
Base Case: $y = [e]$
$head(x \bowtie [e])$

Case $x = [\ ]$
$= head([e])$ [Property of $head$]

$= head([head\ [e]])$ [Definition of *head*]

$= head(x \bowtie [head\ [e]])$ [Property of *head*]

Case $x = \bot$

$= head(\bot)$ [Property of *head*]

$= head(\bot \bowtie [head\ [e]])$ [Property of *head*]

Inductive step on x:

$fs :$ finseq X

$is :$ pfiseq X

We assume that

$head(fs \bowtie y) = head(fs \bowtie [head\ y]), y \neq [\ ]$

$head(is \bowtie y) = head(is \bowtie [head\ y]), y \neq [\ ]$

We must prove that

$head((fs \bowtie is) \bowtie y) = head((fs \bowtie is) \bowtie [head\ y]), y \neq [\ ]$ [Property of *head*]

Inductive step on y:

$gs :$ finseq X

$hs :$ pfiseq X

We assume that

$head(x \bowtie gs) = head(x \bowtie [head\ gs]), gs \neq [\ ]$

$head(x \bowtie hs) = head(x \bowtie [head\ hs]), hs \neq [\ ]$

We must prove that

$head(x \bowtie (gs \bowtie hs)) = head(x \bowtie [head(gs \bowtie hs)]), gs \neq [\ ]$

Case $x = [\ ]$

$= head((gs \bowtie hs))$ [Property of *head*]

$= head([\ ] \bowtie [head\ (gs \bowtie hs)])$ [Property of *head*]

Case $x = \bot$

$= head(\bot)$ [Definition of $\bowtie$/]

$= head(\bot \bowtie [head\ (gs \bowtie hs)])$ [Definition of $\bowtie$/]

Inductive step on x:

$fs :$ finseq X

$is :$ pfiseq X

We assume that

$head(fs \bowtie (gs \bowtie hs)) = head(fs \bowtie [head(gs \bowtie hs)]), (gs \bowtie hs) \neq [\ ]$

$head(is \bowtie (gs \bowtie hs)) = head(is \bowtie [head(gs \bowtie hs)]), (gs \bowtie hs) \neq [\ ]$

We must prove that
$$head((\textit{fs} \bowtie \textit{is}) \bowtie (\textit{gs} \bowtie \textit{hs})) = head((\textit{fs} \bowtie \textit{is}) \bowtie [head(\textit{gs} \bowtie \textit{hs})]), (\textit{gs} \bowtie \textit{hs}) \neq [\,]$$
$$\text{[Property of } head]$$

□

D.6.24 Lemma 28

$$l_1 \bowtie (l_2 \bowtie l_3) = (l_1 \bowtie l_2) \bowtie l_3$$

$l_1 \bowtie (l_2 \bowtie l_3)$

$= \{x : l_1;\ y : l_2 \bowtie l_3 \bullet x \,^\wedge\, y\}$	[Definition of $\bowtie$]
$= \{x : l_1;\ y : \{w : l_2;\ z : l_3 \bullet w \,^\wedge\, z\} \bullet x \,^\wedge\, y\}$	[Definition of $\bowtie$]
$= \{x : l_1;\ w : l_2;\ z : l_3 \bullet x \,^\wedge\, (w \,^\wedge\, z)\}$	[Set Comprehension]
$= \{x : l_1;\ w : l_2;\ z : l_3 \bullet (x \,^\wedge\, w) \,^\wedge\, z\}$	[Lemma D.6.18]
$= \{v : \{x : l_1;\ w : l_2 \bullet (x \,^\wedge\, w)\};\ z : l_3 \bullet v \,^\wedge\, z\}$	[Set Comprehension]
$= \{v : l_1 \bowtie l_2;\ z : l_3 \bullet v \,^\wedge\, z\}$	[Definition of $\bowtie$]
$= (l_1 \bowtie l_2) \bowtie l_3$	[Definition of $\bowtie$]

□

191

Appendix E

Gabriel's Architecture

We present now some diagrams that describes **Gabriel**'s architecture. The whole document can be found in [26].

E.1 Class Diagrams

E.1.1 Integration **Gabriel** – Refine

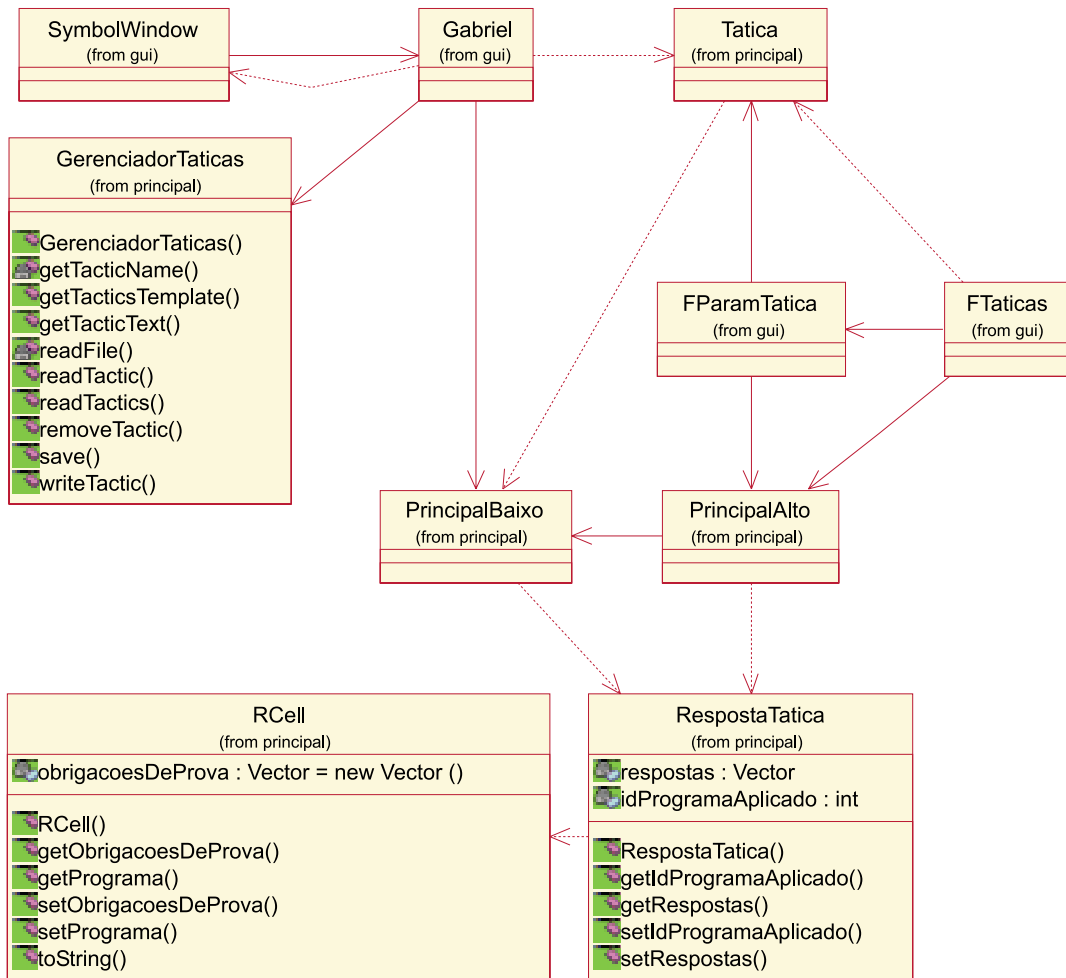This class diagram presents the main classes of **Gabriel** and Refine, and their integration.

Figure E.1: Integration **Gabriel** – Refine

E.1.2   Tactic's Hierarchy

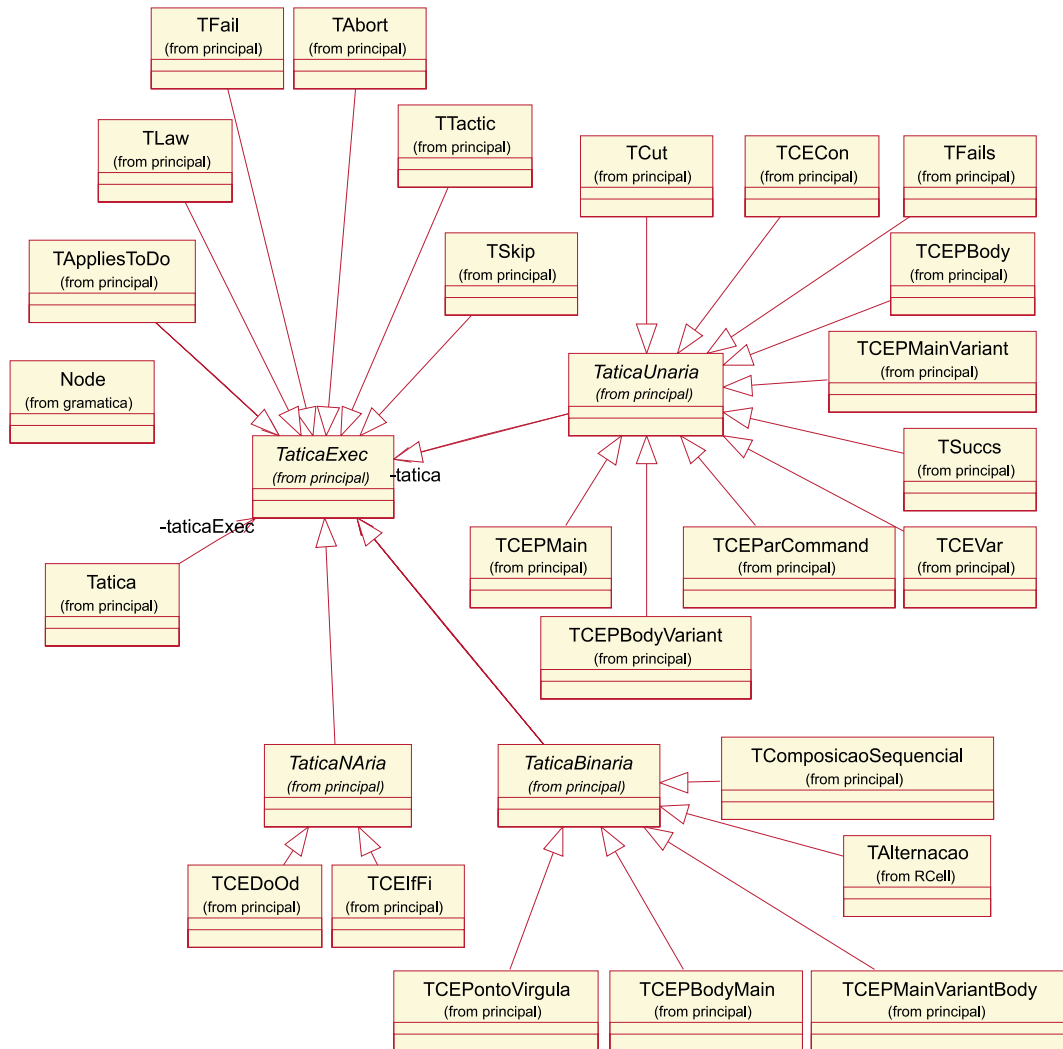This class diagram presents **Gabriel**'s tactics hierarchy .

Figure E.2: Tactic's Hierarchy

E.2 Sequence Diagrams

E.2.1 Tactic Generation

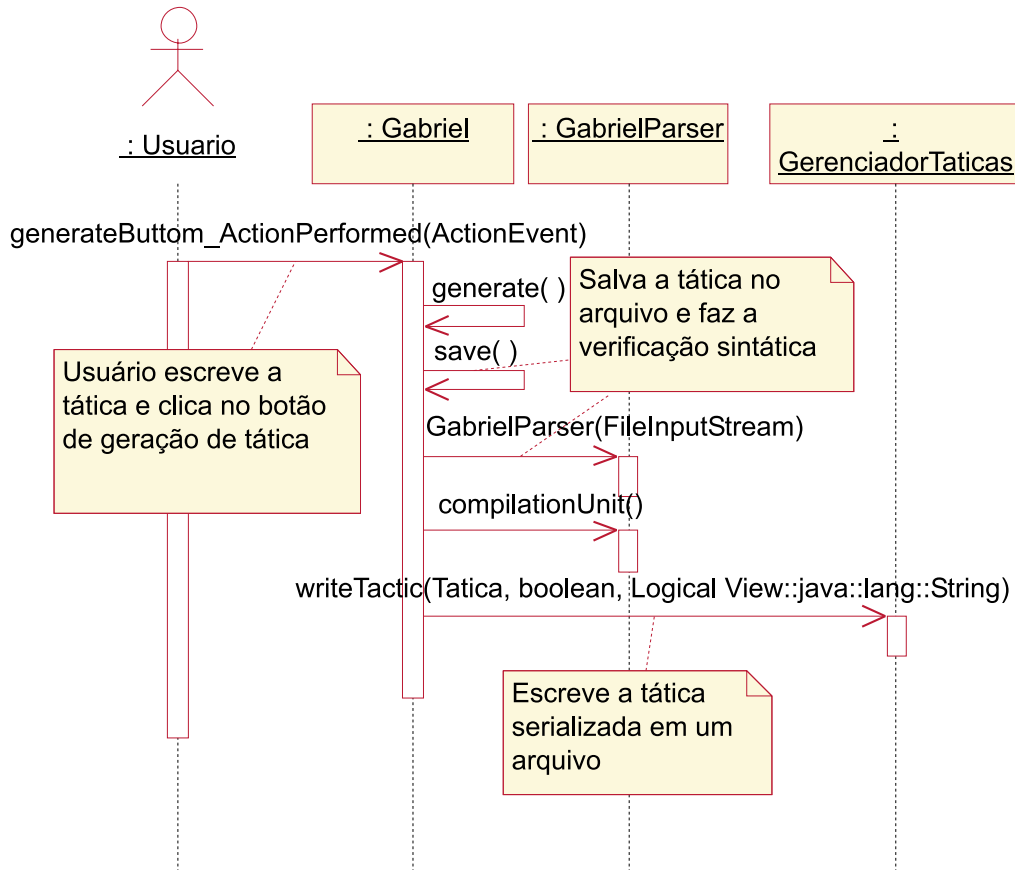This sequence diagram describes the tactic generation in Gabriel.

Figure E.3: Tactic Generation

E.2.2 Tactic Application

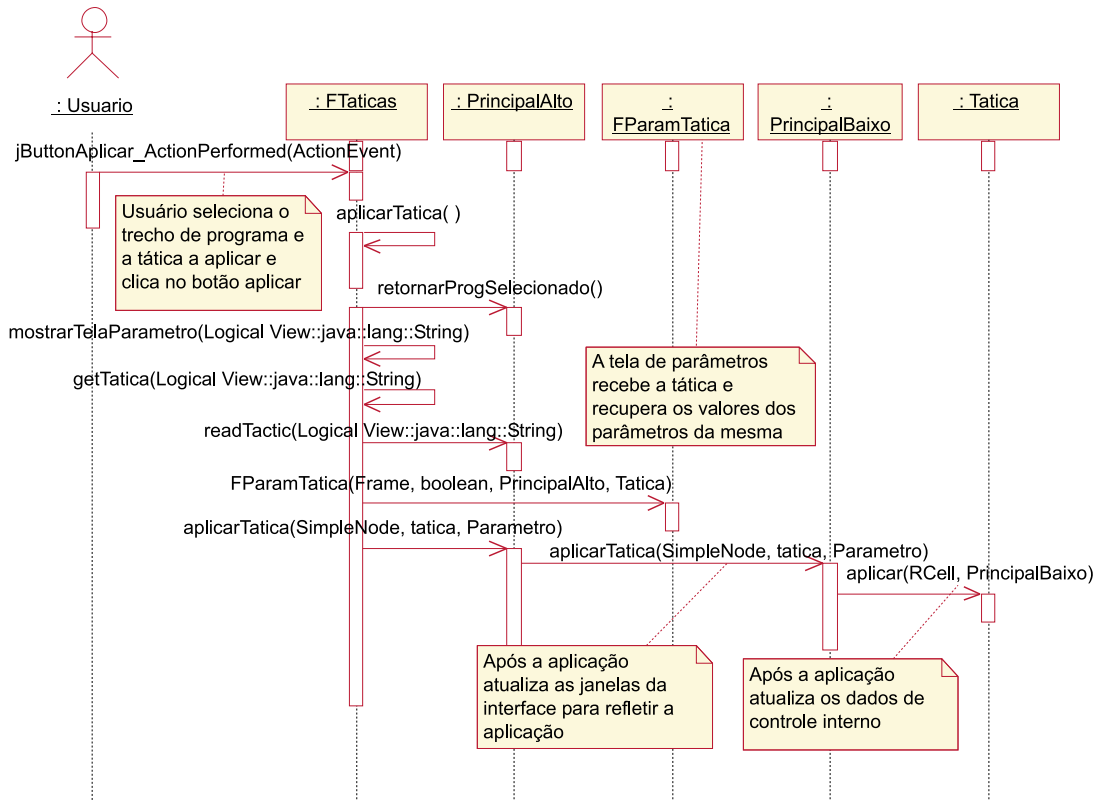This sequence diagram describes the tactic application in Gabriel.

Figure E.4: Tactic Application

Appendix F

Constructs of **Gabriel**

We present now the constructs of Gabriel for ArcAngel.

F.1 ASCII **ArcAngel**

Table F.1 presents the constructs os ArcAngel written in a ASCII representation.

ArcAngel	Gabriel
law	*law*
tactic	*tactic*
skip	*skip*
fail	*fail*
abort	*abort*
;	;
|	|
!	!
succs	*succs*
fails	*fails*
⸨;⸩	|; |
if **fi**	| *if* | | *fi* |
do **od**	| *do* | | *od* |
var ⟧	| *var* | |] ||
con ⟧	| *con* | |] ||
pmain ⟧	| *pmain* | |] ||
pmainvariant ⟧	| *pmainvariant* | |] ||
pbody ⟧	| *pbody* | |] ||
pbodyvariant ⟧	| *pbodyvariant* | |] ||
pbodymain ⟧	| *pbodymain* | |] ||
pmainvariantbody ⟧	| *pmainvariantbody* | |] ||
parcommand ⟧	| *parcommand* | |] ||
applies to do	*applies to do*

Table F.1: ArcAngel's constructs in Gabriel

F.2 Laws Names and Templates

Tables F.2 and F.3 present the law names and their usage templates in Gabriel. The usage template defines the name of the laws and their arguments.

Morgan's Refinement Calculus	Gabriel
Law 1.1	$strPost(PRED(newPost))$
Law 1.2	$weakPre(PRED(newPre))$
Law 1.3	$assign(IDS(lstVar), EXPS(lstVal))$
Law 1.7	$simpleEspec()$
Law 1.8	$absAssump()$
Law 3.2	$skipIntro()$
Law 3.3	$seqComp(PRED(mid))$
Law 3.4	$skipComp()$
Law 3.5	$fassign(IDS(lstVar), EXPS(lstVal))$
Law 4.1	$alt(PREDS(guards))$
Law 4.3	$altGuards(PREDS(guards))$
Law 5.1	$strPostIV(PRED(newPost))$
Law 5.2	$assigIV(IDS(lstVar), EXPS(lstVal))$
Law 5.3	$skipIntroIV()$
Law 5.4	$contractFrame(IDS(vars))$
Law 5.5	$iter(PREDS(guard), EXP(variant))$
Law 6.1	$varInt(DECS(newV : T))$
Law 6.2	$conInt(DECS(newC), PRED(pred))$
Law 6.3	$fixInitValue(DECS(newC), EXPS(exps))$
Law 6.4	$removeCon(IDS(cons))$
Law 8.3	$expandFrame(IDS(newVars))$
Law 8.4	$seqCompCon(PRED(mid), DECS(conDecs), IDS(lstVar))$
Law B.2	$seqCompIV(PRED(mid), IDS(lstVar))$

Table F.2: Gabriel's Laws Names

Morgan's Refinement Calculus	Gabriel
Law 11.1	$procNoArgsIntro(EXP(name), PROG(body))$
Law 11.2	$procNoArgsCall()$
Law 11.3	$procArgsIntro(EXP(name), PROG(body),$ $PARAMS(resx:Z))$
Law 11.4	$procArgsCall()$
Law 11.5	$callByValue()$
Law 11.6	$callByValueIV()$
Law 11.7	$callByResult(DECS(newVar), IDS(subsVar))$
Law 11.8	$multiArgs()$
Law 11.9	$callByValueResult()$
Law 13.1	$variantIntro(EXP(name), PROG(body),$ $IDS(varName), EXP(varExp),$ $PARAMS(resx:Z))$
Law 13.2	$procVariantBlockCall()$
Law 13.3	$recursiveCall()$
Law 13.4	$variantNoArgsIntro(EXP(name), PROG(body),$ $IDS(varName), EXP(varExp))$
Law 13.5	$procVariantBlockCallNoArgs()$
Law 13.6	$recursiveCallNoArgs()$
Law 17.1	$coercion()$
Law 17.2	$absCoercion()$
Law 17.3	$intCoercion(PRED(post))$

Table F.3: Continuation of Gabriel's Laws Names

F.3 Argument Types

Table F.4 presents argument types in Gabriel.

Type Description	Gabriel's Representation	Example
Predicate	$PRED(pred)$	$PRED(x > 0)$
Expression	$EXP(exp)$	$EXP(x + 1)$
Program	$PROG(prog)$	$PROG(x : [x >= 0, x = x + 1])$
List of Declarations	$DECS(dec_1; \ ...; \ dec_n)$	$DECS(x : T; \ y : S)$
List of Predicates	$PREDS(pred_1, ..., pred_n)$	$PREDS(x > 0, x <= 0)$
List of Expressions	$EXPS(exp_1, ..., exp_n)$	$EXPS(x + 1, x - 1)$
List of Identifiers	$IDS(var_1, ..., var_n)$	$IDS(x, y, z)$
List of Parameters	$PARAMS(par_1; \ ...; \ par_n)$	$PARAMS(res \ a; \ val \ b)$

Table F.4: Argument's Types in Gabriel

Bibliography

[1] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29:841–862, 1982.

[2] R. J. R. Back. Procedural Abstraction in the Refinement Calculus. Technical report, Department of Computer Science, Åbo - Finland, 1987. Ser. A No. 55.

[3] R. J. R. Back and J. von Wright. Refinement Calculus, Part I: Sequential Non-deterministic Programs. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 42 – 66, Mook, The Netherlands, 1989. Springer-Verlag.

[4] R. J. R. Back and J. von Wright. Refinement Concepts Formalised in Higher Order Logic. *Formal Aspects of Computing*, 2:247–274, 1990.

[5] Holger Becht, Anthony Bloesch, Ray Nickson, and Mark Utting. Ergo 4.1 reference manual.

[6] D. Carrington, I. Hayes, R. Nickson, G. Watson, and J. Welsh. A program refinement tool. *Formal Aspects of Computing*, 10(2):97–124, 1998.

[7] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[8] S. L. Coutinho, T. P. C. Reis, and A. L. C. Cavalcanti. Uma Ferramenta Educacional de Refinamentos. In *XIII Simpósio Brasileiro de Engenharia de Software*, pages 61 – 64, Florianópolis - SC, 1999. Sessão de Ferramentas.

[9] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[10] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[11] L. Groves, R. Nickson, and M. Utting. A Tactic Driven Refinement Tool. In C. B. Jones, R. C. Shaw, and T. Denvir, editors, *5th Refinement Workshop*, Workshops in Computing, pages 272 – 297. Springer-Verlag, 1992.

[12] Lindsay Groves. Adapting formal derivations. Technical Report 1995.CS-TR-95-9, 1995.

[13] Lindsay Groves. Deriving programs by combining and adapting refinement scripts. Technical Report 1995.CS-TR-95-13, 1995.

[14] J. Grundy. A Window Inference Tool for Refinement. In C. B. Jones, R. C. Shaw, and T. Denvir, editors, *5th Refinement Workshop*, Workshops in Computing, pages 230 – 254. Springer-Verlag, 1992.

[15] C. A. R. Hoare, I. J. Hayes, He Jifeng, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey, and B. A. Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.

[16] Inc. John Wiley & Sons, editor. *The Elements of User Interface Design*. Springer Verlag, 1997.

[17] A. Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice-Hall, 1990.

[18] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 2nd edition, 1987.

[19] A. Martin. Infinite Lists for Specifying Functional Programs in Z. Technical report, University of Queensland, Queensland - Australia, March 1995.

[20] A. P. Martin. *Machine-Assisted Theorem Proving for Software Engineering*. PhD thesis, Oxford University Computing Laboratory, Oxford, UK, 1996. Technical Monograph TM-PRG-121.

[21] A. P. Martin, P. H. B. Gardiner, and J. C. P. Woodcock. A Tactical Calculus. *Formal Aspects of Computing*, 8(4):479–489, 1996.

[22] A. J. R. G. Milner. Is Computing an Experimental Science? Technical Report ECS-LFCS-86-1, University of Edinburgh, Department of Computer Science, Edinburgh - UK, August 1986.

[23] Carroll Morgan. *Programming from Specifications*. Prentice-Hall, 2nd edition, 1994.

[24] M. V. M. Oliveira. Teste de Usabilidade de REFINE e T - REFINE. Technical report, Centro de Informática - Universidade Federal de Pernambuco, Pernambuco - Brazil, December 2001. At http://www.cin.ufpe.br/~mvmo/gabriel/.

[25] M. V. M. Oliveira. ArcAngel: Tactics Examples and Their Usage. Technical report, Centro de Informática - Universidade Federal de Pernambuco, Pernambuco - Brazil, December 2002. At http://www.cin.ufpe.br/~mvmo/gabriel/.

[26] M. V. M. Oliveira. Gabriel's Rose Model. Rose model, Centro de Informática - Universidade Federal de Pernambuco, Pernambuco - Brazil, December 2002. At http://www.cin.ufpe.br/~mvmo/gabriel/.

[27] M. V. M. Oliveira. *Refine-Gabriel Project Page - Documents and Downloads*, 2002. At http://www.cin.ufpe.br/~mvmo/gabriel/.

[28] M. V. M. Oliveira and A. L. C. Cavalcanti. Tactics of refinement. In *14th Brazilian Symposium on Software Engineering*, pages 117 – 132, 2000.

[29] Peter J. Robinson and John Staples. Formalizing the Hierarchical Structure of Practical Mathematical Reasoning. *Journal of Logic and Computation*, 3(1):47–61, 1993.

[30] A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. Technical Report Programming Research Group Technical Monograph PRG-53, 1986.

[31] Jan L. A. van de Snepscheut. Proxac: An editor for program transformation. Technical Report 1993.cs-tr-93-33, 1993.

[32] Jan L. A. van de Snepscheut. Mechanised support for stepwise refinement. In Jürg Gutknecht, editor, *Programming Languages and System Archtectures*, volume 782 of *Lecture Notes in Computer Science*, pages 35–48. Springer, March 1994. Zurich, Switzerland.

[33] T. Vickers. An Overview of a Refinement Editor. In *5th Australian Software Engineering Conference*, pages 39–44, Sydney - Australia, May 1990.

[34] T. Vickers. A language of refinements. Technical Report TR-CS-94-05, Computer Science Department, Australian National University, 1994.

[35] J. von Wright. Program Refinement by Theorem Prover. In D. Till, editor, *6th Refinement Workshop*, Workshops in Computing, pages 121 – 150, London - UK, 1994. Springer-Verlag.

[36] J. von Wright, J. Hekanaho, P. Luostarinen, and T. Långbacka. Mechanizing Some Advanced Refinement Concepts. *Formal Methods in System Design*, 3:49–81, 1993.