Tactics of Refinement

Marcel Oliveira
Centro de Informática/UFPE
PO Box 7851 50740-540 Recife PE Brazil
{mvmo}@cin.ufpe.br

Abstract

The refinement calculus is a modern technique of formal program development. Its application, however, may lead to long and repetitive developments. In this monograph we present a language to write refinement tactics and present examples of useful tactics. They encompass the application of several refinement laws, but can be used as a single transformation rule. We specify the semantics of the language formally and define a normal form for tactics written in a subset of this language. Using tactics is not a novel idea, but apparently, in the context of refinement the only existing work uses Prolog as a tactic language. Our language does not depend of any programming language or tool. Also, we are not aware of any presentation of refinement strategies written in the form of tactics as we present here.

Keywords: Formal methods, program development, refinement calculus, tactic language.

1 Introduction

Morgan's refinement calculus [10] 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 monograph we present RTL (Refinement Tactic Language), a language for the definition of refinement tactics based on Angel [9]. This is a general tactic language that is not tailored to any particular proof tool. It makes no assumption about the form of proof goals and the rules that are applied to them. Moreover, the semantics of Angel is well-defined and it has an associated algebraic theory that allows the proof of properties of tactics.

The proof of theorems is usually based on inference rules that involve only terms of the logic. Therefore, Angel assumes that rules transform proof goals into proof goals. For refinement tactics we need a language that takes into account the fact that refinement laws transform programs into programs, but generate proof obligations. These are not affected by later applications of refinement laws, but the result of applying a sequence of laws (tactic) is a program and the set of all proof obligations generated by the individual law applications.

The constructs of RTL are similar to those of Angel, but are adapted to deal with refinement laws and programs. Moreover, RTL provides structural combinators that are suitable to apply refinement laws to components of programs. Using RTL, we define refinement tactics that embody commom development and programming strategies [10, 8, 2].

The semantics of RTL is also similar to Angel's semantics, but the notion of goal is elaborated to take account the particularities of the refinement calculus. Most of the algebraic laws in [9] are not proved. In this monograph we present their proofs based on the RTL semantics. A normal form is also presented in this monograph, it is similar to that presented in [9] for angel tactics. Our contribution is to give more details on the proofs of the lemmas and therorems involved in the strategy of reduction to this normal form.

The use of tactics to guide proofs is not a novel idea [9]. Many proof tools provide tactic languages to write programs that help in the construction of proofs [6, 11]. As far as we know, however, refinement tactics have only been considered in [7], where Prolog is used as a tactic language.

In Section 2 we give an overview of the refinement calculus. Section 3 introduces RTL. In Section 4 we present the semantic model of RTL. Section 5 shows simple laws for reasoning about the language. Section 7 presents a normal form for tactics written in a subset of RTL. In Setion 8 we present some tactics and examples of their applications. Finally in Section 9 we discuss related and future works.

2 Refinement Calculus

The refinement calculus is based on a unified language of specification, design and implementation. There is no difference between specification and programs. Developing programs consists of applying refinement laws to a specification repeatedly until an adequate program is obtained. Some of these laws are listed in Appendix B.

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.

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.

Besides the specification statement, the language of the refinement calculus includes all the constructors of Dijkstra's language [5]. 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 arguments declarations, if the procedure has arguments. Then, after a $\bullet$ the main program is declared. The general form is $[\![\mathbf{proc}\ name = body \bullet prog]\!]$. The arguments can be passed by value, using the keyword **val**, by result, keyword **res**, or by value-result, using the keyword **val_res**. An example of a parameterized procedure is $[\![$**proc** $inc = ($**val_res** $n : \mathbb{N} \bullet n := n + 1) \bullet inc(n)]\!]$. The body of the procedure in this case is a parameterized command [1].

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 $[\![$**proc** $name = body$ **variant** v **is** $e \bullet prog]\!]$.

3 RTL

The simplest form of a tactic is a simple law application: **law** $Name(\{argument\}^*)$. When applying a law to a program, there are two possible outcomes: if the law is applicable to the program the law is actually applied and changes the program, possibly generating proof obligations; if the law is not applicable to the program, the application of the law fails.

Sometimes it is very useful to apply a previously defined tactic in another tactic. In RTL we can do this using the construct **tactic** $tacticName(\{argument\}^*)$. The behavior of this tactic is similar to that of a **law** tactic, except that it applies a whole tactic and not a single law. The distinction is useful for documentation purposes.

Two special tactics are introduced to help us to define some tactics; they are **skip** and **fail**. The first one always succeeds, does not change the program, and also does not generate proof obligations. The second one always fails.

3.1 Tacticals

In RTL tactics can be combined in sequence. The sequential composition of two tactics is written $t_1\,;\,t_2$. This tactics 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 this tactic fails. The proof obligations generated by the application of this tactic are those resulting from the application of t_1 and t_2.

For example, if we apply the tactic **law strPost**$(x = 10)$; **law weakPre**$(true)$ to the program $x : [x < 10, x >= 10]$, first we have the application of **strPost**$(x = 10)$. This is the strengthen postcondition refinement law, which takes a new postcondition as argument. The result is $x : [x < 10, x = 10]$ and the proof obligation is $x = 10 \Rightarrow x \geq 10$. Then, **weakPre**$(true)$, the weaken precondition law with parameter $true$, is applied to $x : [x < 10, x = 10]$, and we get $x : [x = 10]$ with proof obligations $x = 10 \Rightarrow x \geq 10$ and $x < 10 \Rightarrow \mathrm{true}$.

Tactics can also be combined in alternation, which is written $t_1 \mid t_2$. This tactic applies t_1 to the program. If the application of t_1 succeeds, then this tactic succeeds, else this tactic applies t_2 to the program. If the application of t_2 succeeds then this tactic succeeds, else the whole tactic fails. When a tactic presents such a choice of next steps, the one that leads to success, if any, is chosen. This is implemented using backtracking.

Consider, by way of illustration, applying to the specification statement $x : [x < 10, x \geq 10]$ the tactic (**law assig**$(< x >, < 10 >)$ $\mid$ **law strPost**$(x = 10)$); **law weakPre**$(true)$. First, this tactic applies **assig**$(< x >, < 10 >)$, the assignment introduction law, to the given program. This application succeeds and results in the assignment $x := 10$ with the proof obligation $x < 10 \Rightarrow 10 = 10$. Then, the tactic tries to apply **weakPre**$(true)$ to $x := 10$. This application fails because **weakPre** only applies to specification statements. Backtracking occurs to consider the second branch of the alternation: **strPost**$(x = 10)$. This application succeeds and we get $x : [x < 10, x = 10]$ with the proof obligation $x = 10 \Rightarrow x \geq 10$. Finally, the tactic tries to apply **law weakPre**$(true)$ to $x : [x < 10, x = 10]$. This application succeeds and we get the program $x : [x = 10]$ with proof obligations $x = 10 \Rightarrow x \geq 10$ and $x < 10 \Rightarrow$ true. If this last law application had failed then the whole tactic would have failed as well.

This kind of behavior may lead to problems of inefficient searches. So, RTL includes the cut operator (!). The tactic $!\,t$ behaves just like t: it returns the first successful tactic application; if a subsequent tactic application fails, however, then the whole tactic fails. For example, the application of !(**law assig**$(< x >, < 10 >)$ | **law strPost**$(x = 10)$); **law weakPre**$(true)$ to the program $x : [x < 10, x \geq 10]$ fails since the application of **weakPre** to $x := 10$ fails.

We also have the recursion operator μ. As an example we have the tactic *exhaust*. When applied to a tactic t, this tactic applies t as many times as possible, terminating with success when the application of t fails. Its definition is $exhaust\ t = (\mu\, Y \bullet (t;\ Y \mid \mathbf{skip}))$.

Sometimes the recursive application of a tactic generates a situation where it neither fails nor succeeds, but keeps running indefinitely. To reason about this kind of problem, it is necessary to introduce a tactic which presents this behavior, this is **abort**.

When we want to define the programs to which the tactic can be applied we use the tactic **applies to** p **do** *tactic*, which introduces a meta-program p that characterizes the program to which this tactic is applicable. The meta-variables used in p can be used in *tactic*. Meta-programs are programs written in a general format. As an example we have the meta-program $w : [pre, post]$ which has as its meta-variables $w, pre, post$. By way of illustration, we consider **applies to** $w : [pre_1 \wedge pre_2, post]$ **do law weakPre**(pre_1); **law strPost**$(post \wedge pre_2)$, which applies only to specifications of the form $w : [pre_1 \wedge pre_2, post]$.

3.2 Structural Combinators

In some cases the program has subprograms and we want to apply tactics to each of them. This is made using *structural combinators*. For example the tactic $t_1 \boxed{;} t_2$ 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 by the application of this tactic are those generated by the application of t_1 to p_1 and of t_2 to p_2.

In the case of an alternation, we use the combinator $\boxed{\mathbf{if}}\ t_1, ..., t_n\ \boxed{\mathbf{fi}}$. When applied to an alternation $\mathbf{if}\ g_1 \rightarrow p_1 \,[]\, ... \,[]\, g_n \rightarrow p_n\ \mathbf{fi}$, it applies each of the tactics to the corresponding program. For example, if we apply to $\mathbf{if}\ x \geq 0 \rightarrow x : [x \geq 0] \,[]\, x < 0 \rightarrow x : [x < 0]\ \mathbf{fi}$ the tactic $\boxed{\mathbf{if}} < \mathbf{law\ assig}(< x >, < 1 >), \mathbf{law\ assig}(< x >, < -1 >) > \boxed{\mathbf{fi}}$, we get the program $\mathbf{if}\ x \geq 0 \rightarrow x := 1 \,[]\, x < 0 \rightarrow x := -1\ \mathbf{fi}$. The resulting proof obligations are true $\Rightarrow 1 \geq 0$ and true $\Rightarrow -1 < 0$. For iterations $\mathbf{do}\ g_1 \rightarrow p_1 \,[]\, ... \,[]\, p_n \rightarrow p_n\ \mathbf{od}$ we have a similar structural combinator $\boxed{\mathbf{do}}\, t_1, ..., t_n\ \boxed{\mathbf{od}}$.

In the case we have a variable block we use the structural combinator $\boxed{\mathbf{var}}\ t\ \boxed{]\!]}$, which applies the tactic t to the body of the block. For example, if we apply $\boxed{\mathbf{var}}$ **law strPost**$(x > 0)\boxed{]\!]}$ to $[\![\mathbf{var}\ x : \mathbb{N} \bullet x : [x \geq 0]]\!]$ we get $[\![\mathbf{var}\ x : \mathbb{N} \bullet x : [x > 0]]\!]$ and the proof obligation $x > 0 \Rightarrow x \geq 0$. Similarly, we have the structural combinator $\boxed{\mathbf{cons}}\ t\ \boxed{]\!]}$, in the case we have a logical constant block. This structural combinator also applies the tactic t to the body of the block.

The structural combinators $\boxed{\mathbf{pmain}}\ t\ \boxed{]\!]}$ and $\boxed{\mathbf{pmainvariant}}\ t\ \boxed{]\!]}$ are applied to procedure blocks and variant blocks, respectively. They apply t to the main program of procedure and variant blocks, respectively. For example, if we have the tactic $\boxed{\textbf{pmain}}$ **law strPost**$(x > 0)\boxed{\rrbracket}$ and we apply this tactic to the procedure block $\llbracket$**proc** $nonNeg = x : [x > 0] \bullet x : [x \geq 0]\rrbracket$ we get $\llbracket$**proc** $p = x : [x > 0] \bullet x : [x > 0]\rrbracket$ and the proof obligation $x > 0 \Rightarrow x \geq 0$.

When we want to apply a tactic to a procedure body we use the structural combinators $\boxed{\textbf{pbody}}\, t \boxed{\rrbracket}$ and $\boxed{\textbf{pbodyvariant}}\, t \boxed{\rrbracket}$, which apply to procedure blocks and variant blocks, respectively. For example, if we apply the tactic $\boxed{\textbf{pbody}}$ **law assig**$(< x >, < 10 >)\boxed{\rrbracket}$ to the program $\llbracket$**proc** $nonNeg = x : [x > 0] \bullet nonNeg\rrbracket$, we get $\llbracket$**proc** $nonNeg = x := 10 \bullet nonNeg\rrbracket$ and the proof obligation $true \Rightarrow 10 \geq 0$.

In the case we have arguments declaration, we use the structural combinators $\boxed{\textbf{val}}\ t$, $\boxed{\textbf{res}}\ t$, and $\boxed{\textbf{val\_res}}\, t$ depending on whether the arguments are passed by value, result, or value-result, respectively. For example, if we apply the tactic $\boxed{\textbf{pbody}}\ \boxed{\textbf{val\_res}}$ **law strPost**$(x > 0)\boxed{\rrbracket}$ to the procedure block $\llbracket$**proc** $nonNegArg = ($**val_res** $x : \mathbb{N} \bullet x : [x \geq 0]) \bullet x : [x > 0]\ \rrbracket$, we get the program $\llbracket$**proc** $nonNegArg = ($**val_res** $x : \mathbb{N} \bullet x : [x > 0]) \bullet x : [x > 0]\ \rrbracket$ and the proof obligation $x > 0 \Rightarrow x \geq 0$.

The declaration of a tactic has the form

> **Tactic** *name* (*args*) *tactic*
> [**proof obligations** $\{predicate\}^*$] [**program generated** *program*] **end**

where *name* is the name of the tactic being defined, and *args* are its arguments. For documentation purposes, we include clauses **proof obligations** and **program generated**; the former lists the proof obligations generated by the application of *tactic*, and the latter presents the program generated. These two clauses are optional as this information can be infered from the tactic itself. In the next section we give several examples of tactics declarations.

4 Semantics of RTL

4.1 Abstrat Syntax

RTL has a very simple syntax which can be defined using BNF-notation as follows.

tactic ::= **law** $name(\{argument\}^*)$ |

skip |

fail |

tactic $name(\{argument\}^*)$ |

tactic ; *tactic* |

tactic | *tactic* |

! *tactic* |

μ *variable* $\bullet$ *tactic* $|$

abort $|$

tactic $\boxed{;}$ *tactic* $|$

$\boxed{\textbf{if}}$ $\{tactic\}^+$ $\boxed{\textbf{fi}}$ $|$

$\boxed{\textbf{do}}$ $\{tactic\}^+$ $\boxed{\textbf{od}}$ $|$

$\boxed{\textbf{var}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{cons}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{pmain}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{pmainvariant}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{pbody}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{pbodyvariant}}$ *tactic* $\boxed{]\!]}$ $|$

$\boxed{\textbf{val}}$ *tactic* $|$

$\boxed{\textbf{res}}$ *tactic* $|$

$\boxed{\textbf{val\_res}}$ *tactic* $|$

applies to *program* **do** *tactic*

tacticDecl ::= **Tactic** *name* ($\{argument\}^*$) *tactic*
[**proof obligations** $\{predicate\}^*$]
[**program generated** *program*]
tacticProg ::= $\{tacticDecl\}^*$ *tactic*

argument ::= *name* $|$
predicate $|$
expression $|$
declaration

The syntatic category *name* contains all valid law and tactic names. The syntatic category *predicate* is that of the first order logic formulas; *expression* is the set of expressions involving operators over data types as sets and lists; *declaration* contains the variable and constant declarations; and *program* is the set of programs of the refinement calculus. Their definitions are standard and omitted.

A tactic program consists of

$tacticProg ::= \{tacticDecl\}^* \ tactic$

Usually the tactic makes use of the declared tactics.

4.2 Formal Definition of RTL Operators

In this section we give a formal semantics to the RTL constructs using a denotational style.

Formally, a tactic is applied to a structure called *RMCell* (Refinement Management Cell). This is a pair which tipically contains the program to which the tactic is to be applied, and a set of proof obligations produced by the derivation of this program.

$$RMCell = (program, \textbf{set of } predicate)$$

The result of a tactic aplication is a list of *RMCells* which contains all the possible outcomes of the aplication of the tactic: all the possible programs it can generate with corresponding proof obligations. These include the proof obligations in the *RMCell* to which the tactic was applied and those generated by the tactic aplication. If the aplication of some tactic to the *RMCell* fails, then the empty list is returned.

$$Tactic = RMCell \nrightarrow RMCell*$$

For a set A, $A*$ is the set of all lists, whose elements are drawn from A.

To give the semantics of the contructs **law** $name(\{argument\}^*)$ and **tactic** $name(\{argument\}^*)$ we need a *laws environment* and a *tactics environment*, respectively. These two environments are functions as defined below:

$$lawsenvironment :: name \rightarrow \{argument\}^* \rightarrow program \rightarrow RMCell$$

$$tacticsenvironment :: name \rightarrow \{argument\}^* \rightarrow Tactic$$

The *lawsenvironment* is a function which takes a law *name*, a list of arguments, and a *program* to which the law can be applied, and returns a pair in which the first element is the *program* generated by the aplication of the law to the program, and the second element is the set of proof obligations generated by the same aplication.

For example, if Γ_L is an environment that records the law **strPos**, we have that for a predicate $post'$ and a program $w : [pre, post]$

$$\Gamma_L \ \textbf{strPos} \ < post' > \ w : [pre, post] = (w : [pre, post'], \{post' \Rightarrow post\})$$

This means that if we apply the law **strPos** with argument $post'$ to the specification statement $w : [pre, post]$, we change the postcondition to $post'$, but we must prove that $post' \Rightarrow post$. Typically, we are interested in environments that record at least the laws of the refinement calculus shown in Appendix B.

The *tacticsenvironment* is a function which takes a tactic *name*, a lsit of arguments, and returns a *Tactic*. As an example, we present the tactic **nextAndThisAssig**. This tactic generates two assignments in sequence from a specification. If Γ_T is an environment that records this tactic we have that, for two lists of variables *lstVarNext* and *lstVarThis*, two list of values *lstValNext* and *lstValThis*, and an *RMCell* $(w : [pre, post], proofs)$

$$
\begin{array}{l}
\Gamma_T \ \textbf{nextAndThisAssig} \\
\quad < (lstVarNext, lstValNext, lstVarThis, lstValThis) > \\
\quad (w : [pre, post], proofs) = \\
< (lstVarThis := lstValThis;\ lstVarNext = lstValNext, \\
\ \ proofs \cup \{pre \Rightarrow post[lstValNext/lstVarNext][lstValThis/lstVarThis]\}) >
\end{array}
$$

The arguments *lstVarThis* and *lstValThis* determine the first assignment, where *lstVarThis* is the list of variables and *lstValThis* is the list of values, and the arguments *lstVarNext* and *lstValNext* are used in the second assignment. The proof obligation

$$pre \Rightarrow post[lstValNext/lstVarNext][lstValThis/lstVarThis]$$

is added to the set of already existing proof obligations as a result of applying this law.

All the semantic funtions in this document use these environments to map the law(tactic) names to *Laws*(*Tactics*). The first semantic function we define is for tactics.

$$[\![\_]\!]_{\_\ \_\ \_} :: tactic \rightarrow lawsenvironment \rightarrow tacticsenvironment \rightarrow Tactic$$

The basic tactic is that which applies a simple law to an *RMCell*

$$
\begin{array}{l}
[\![\textbf{law}\ \ lname(args)]\!]\ \Gamma_L\ \Gamma_T(p, proofs) \\
\quad if\ \ lname \in \operatorname{dom}\Gamma_L \wedge args \in \operatorname{dom}\Gamma_L\ lname \wedge p \in \operatorname{dom}\Gamma_L\ lname\ \ args\ \ then \\
\quad\quad let\ \ (newp, nproofs) = \Gamma_L\ lname\ \ args\ \ p\ \ in \\
\quad\quad < (newp, proofs \cup nproofs) > \\
\quad else \\
\quad\quad <>
\end{array}
$$

This tactic verifies if *lname* belongs to the domain of the law environment Γ_L, if *args* belongs to the domain of Γ_L *lname*, and if p belongs to the domain of Γ_L *lname args*. If these conditions hold then this tactic succeeds, and returns a new *RMCell* which has, as program, the result of applying the law to the program of the *RMCell* given as argument and, as proof obligations, the union of the set of proof obligations of the *RMCell* given as argument with the set proof obligations generated by the application of this law. If the conditions do not hold, the tactic fails.

The application of the tactic **skip** to an *RMCell*, returns a list with one element: the *RMCell* given as argument. The application of the tactic **fail** to an *RMCell* always fails.

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

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

The semantics of **tactic**($args$) has a similar definition to that of the **law** construct.

$$
\begin{array}{l}
[\![\mathbf{tactic}\ tname(args)]\!]\Gamma_L\ \Gamma_T\ RMCell = \\
\quad if\ tname \in \operatorname{dom}\Gamma_T \wedge args \in \operatorname{dom}\Gamma_T\ tname\ then \\
\quad\quad \Gamma_T\ tname\ args\ RMCell \\
\quad else \\
\quad\quad <>
\end{array}
$$

This tactic verifies if $tname$ belongs to the domain of the tactic environment Γ_T and if $args$ belongs to the domain of $\Gamma_T\ tname$. If these conditions hold then this tactic succeeds, and returns the application of the tactic to the $RMCell$ given as argument. If the conditions do not hold, the tactic fails.

The idea for the sequence operator is first to apply $[\![t_1]\!]\Gamma_L\ \Gamma_T$ to the $RMCell$. This returns a list of $RMCells$. Then, we apply $[\![t_2]\!]\Gamma_L\ \Gamma_T$ to each of the members os the resulting list. This returns a list of lists of $RMCells$. Finally, we use $\frown/$ so that the tactic returns a list of $RMCells$. Formally, the sequence (;) is defined

$$[\![t_1\ ;\ t_2]\!]\Gamma_L\ \Gamma_T = \frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$$

For a total function $f : A \to B$, $f* : A* \to B*$ is the function that operates on lists by applying f to each of their elements; the operator $\bullet$ is used to compose functions, and finally $\frown/$ is the distributed concatenation operation. The formal definition of these operators and the others to follow can be found in Appendix A.

The idea for the alternation operator is to return all the possible outcomes from the application of t_1 and t_2 to the $RMCell$. We apply $[\![t_1]\!]\Gamma_L\ \Gamma_T$ and $[\![t_2]\!]\Gamma_L\ \Gamma_T$ to the $RMCell$. We get two lists of $RMCells$. These lists are the result of the application of t_1 and t_2 to the $RMCell$. What we want as the result of $t_1 \mid t_2$ is the concatenation of these two lists. Formally , the alternation ($|$) is defined

$$[\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T = \frown/ \bullet < ([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![t_2]\!]\Gamma_L\ \Gamma_T) >^{\circ}$$

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

The cut operator is defined as

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

since we want just the first result the tactic returns. The function $head'$ takes a list as argument and returns a list containing just the first element of the argument list; or returns the empty list if the argument list is empty.

For the recursion operator we use a standard definition. We need the definition of infinite lists, and tactics should be functions from $RMCell$ to possibly infinite lists of $RMCells$. All the definitions used before, as $*$ and $\frown/$ should be extended. The recursion operator μ is defined as a least fixed point. For a function f from tactics to tactics, we have that

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

This definition makes sense if the set of tactics is a complete lattice. First, we define a partial order for lists. $\bot_A$ is the least-defined element in the set. It is a partial list, as is any list which ends with $\bot_A$. One list is less than another ($s_1 \sqsubseteq_\infty s_2$) whenever they are equal, or the first is a partial list which forms an initial subsequence of the second. For the set of lists of *RMCells* we have the bottom element $\bot_{RMCell^*}$. Formally, an infinite list is a limit of a suitable (i.e. directed) set of partial lists (the limit of the set of lists S is denoted by $(\bigsqcup_\infty S)$).

For tactics, as they are partial functions from *RMCell* to *RMCell**, we may lift the ordering on lists of *RMCells* to an ordering on tactics. We can say that

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

We can see in [4] that, as the set of lists of *RMCells* is a complete lattice, then the set of tactics is also a complete lattice, using the order above.

The **abort** is a tactic that runs indefinitely. Its semantics is

$$[\![\mathbf{abort}]\!]\Gamma_L\ \Gamma_T = \bot_{RMCell^*}$$

where $\bot_{RMCell^*}$ is the error list of *RMCells*. This is defined as the least-defined element in the set *RMCell**.

Even though we have presented the recursion and **abort** definitions, which use the infinite lists model, we consider only finite lists in the proofs of the laws and the lemmas. The extension of this result is left as future work.

As already said, the structural combinators are used when we need to apply a tactic to parts of a program of a particular form. If the program matches the form of the structural combinator, the tactic is applied, otherwise the aplication of the tactic fails.

First of all, we have the structural combinator $\boxed{;}$ which is defined

$$\begin{array}{l} [\![t_1 \boxed{;} t_2]\!]\Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs) = \\ \qquad \Omega_{sc} * (\prod(<([\![t_1]\!]\Gamma_L\ \Gamma_T)\ (p_1, proofs), ([\![t_2]\!]\Gamma_L\ \Gamma_T)\ (p_2, proofs)>)) \end{array}$$

The $\prod$ function constructs the distributed cartesian product of a list of lists.

The basic idea of this definition is to first generate the lists of *RMCells* resulting from the aplication of the tactics t_1 to p_1 and t_2 to p_2.Then we make the distributed cartesian product of these lists using the operator $\prod$. Afterwards, we use the function Ω_{sc} to generate the *RMCells* which are returned.

The function Ω_{sc} takes a list of *RMCells*, which has exactly two elements, and returns an *RMCell* which has as its program a sequential composition of the programs of the two *RMCells* of the argument list, and as its proof obligations the union of the proof obligations in the *RMCells*.

$$\begin{array}{l} \Omega_{sc} :: RMCell* \rightarrow RMCell \\ \Omega_{sc} < (p_1, proofs_1), (p_2, proofs_2) >= (p_1;\ p_2, proofs_1 \cup proofs_2) \end{array}$$

The structural combinators $\boxed{\mathbf{if}}\_\boxed{\mathbf{fi}}$ and $\boxed{\mathbf{do}}\_\boxed{\mathbf{od}}$ have more elaborate definitions. To illustrate the definitions we use an example which is followed step by step.

Consider the following *RMCell*

$$(\mathbf{if}\ x \geq 0 \rightarrow x : [x \geq 0, x = x_0 + 1] \ [\!]\ x < 0 \rightarrow x : [x < 0, x = x_0 - 1]\ \mathbf{fi}, \{x \geq 10\})$$

and the aplication of the following structural combinator to this *RMCell*

$$\begin{array}{l} \boxed{\mathbf{if}} < \mathbf{law\ weakPre}(\text{true}) \mid \mathbf{law\ assig2}(< x >, < x + 1 >), \\ \quad \mathbf{law\ assig2}(< x >, < x - 1 >) > \boxed{\mathbf{fi}} \end{array}$$

The idea is first to make the list of *RMCells* which have as programs the programs of the guarded commands, since we are interested in applying tactics to them.

The function *brP* takes a guarded command and returns a list of the programs in it.

$$\begin{array}{l} brP :: GuardedCommand \rightarrow program* \\ brP\ \ G \rightarrow P =< P > \\ brP\ \ G \rightarrow P \ [\!]\ GC = P : (brP\ \ GC) \end{array}$$

In the example we have

$$\begin{array}{l} brP(x \geq 0 \rightarrow x : [x \geq 0, x = x_0 + 1] \ [\!]\ x < 0 \rightarrow x : [x < 0, x = x_0 - 1]) = \\ \quad < x : [x \geq 0, x = x_0 + 1], x : [x < 0, x = x_0 - 1] > \end{array}$$

The function Φ_r* takes this list and returns a list of *RMCells* which has as programs the programs of this list and an empty set of proof obligations. The proof obligations are treated later.

$$\begin{array}{l} \Phi_r :: program \rightarrow RMCell \\ \Phi_r\ p = (p, \varnothing) \end{array}$$

In the example we get

$$\begin{array}{l} \Phi_r * (< x : [x \geq 0, x = x_0 + 1], x : [x < 0, x = x_0 - 1] >) = \\ \quad < (x : [x \geq 0, x = x_0 + 1], \varnothing), (x : [x < 0, x = x_0 - 1], \varnothing) > \end{array}$$

The next step is to apply each of the tactics of the *tacticsList* to each of the *RMCells*. With this purpose, we use the function *apply* which 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 a list of the results of the aplication of each function of the first list to the corresponding element of the second list.

In our example we have then

$$\begin{array}{l} apply < \mathbf{law\ weakPre}(\text{true}) \mid \mathbf{law\ assig2}(< x >, < x + 1 >), \\ \qquad \mathbf{law\ assig2}(< x >, < x - 1 >) > \\ \quad < (x : [x \geq 0, x = x_0 + 1], \varnothing), (x : [x < 0, x = x_0 - 1], \varnothing) > \\ =< \mathbf{law\ weakPre}(\text{true}) \mid \mathbf{law\ assig2}(< x >, < x + 1 >)\ (x : [x \geq 0, x = x_0 + 1], \varnothing), \\ \quad \mathbf{law\ assig2}(< x >, < x - 1 >)\ (x : [x < 0, x = x_0 - 1], \varnothing) > \\ =<< (x : [x = x_0 + 1], \{x \geq 0 \Rightarrow \text{true}\}), \\ \qquad (x := x + 1, \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1\}) >, \\ \quad < (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >> \end{array}$$

Now, we have to make the distributed cartesian product of this list to get all the possible combinations of the *RMCells* of the first list with the *RMCells* of the second list.

$$
\begin{aligned}
&\prod << (x : [x = x_0 + 1], \{x \geq 0 \Rightarrow \text{true}\}),\\
&\qquad\qquad (x := x + 1, \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1\}) >,\\
&\qquad < (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >>\\
&=<< (x : [x = x_0 + 1], \{x \geq 0 \Rightarrow \text{true}\}),\\
&\qquad\qquad (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >,\\
&\qquad < (x := x + 1, \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1\}),\\
&\qquad\qquad (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >>
\end{aligned}
$$

We also make a list of the guards of the guarded commands. With this purpose we define the function brG as

$$
\begin{aligned}
&brG :: GuardedComands \rightarrow Guard*\\
&brG\ G \rightarrow P =< G >\\
&brG\ G \rightarrow P \,[]\, GC = G : (brG\ GC)
\end{aligned}
$$

In the example we have

$$
\begin{aligned}
&brG(x \geq 0 \rightarrow x : [x \geq 0, x = x_0 + 1] \,[]\, x < 0 \rightarrow x : [x < 0, x = x_0 - 1]) =\\
&\qquad < x \geq 0, x < 0 >
\end{aligned}
$$

Then we combine the list of guards with each element of the list of *RMCells* we get with the distributed cartesian product above.

$$
\begin{aligned}
&mkGC :: Guard* \rightarrow RMCell ** \rightarrow (GuardedCommand, \textbf{set of}\ predicate)*\\
&mkGC\ gs = (appG\ gs)*
\end{aligned}
$$

The function $appG$ takes the list of guards, the list of *RMCells*, and returns a pair, where the first element is a guarded command and the second is a set of proof obligations.

$$
\begin{aligned}
&appG :: Guard* \rightarrow RMCell* \rightarrow (GuardedCommand, \textbf{set of}\ predicate)\\
&appG\ < g >< r >= (g \rightarrow (fst\ r), snd\ r)\\
&appG\ (g : gs)\ (r : rs) = tg\ (g \rightarrow (fst\ r), snd\ r)\ (appG\ gs\ rs)
\end{aligned}
$$

where

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

We have in our example

$$
\begin{aligned}
&mkGC\ < x \geq 0, x < 0 >\\
&\qquad << (x : [x = x_0 + 1], \{x \geq 0 \Rightarrow \text{true}\}),\\
&\qquad\qquad\qquad (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >,\\
&\qquad\qquad < (x := x + 1, \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1\}),\\
&\qquad\qquad\qquad (x := x - 1, \{x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >>\\
&=< (x \geq 0 \rightarrow x : [x = x_0 + 1] [] \, x < 0 \rightarrow x := x - 1,\\
&\qquad \{x \geq 0 \Rightarrow \text{true}, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}),\\
&\qquad (x \geq 0 \rightarrow x = x + 1 [] \, x < 0 \rightarrow x := x - 1,\\
&\qquad \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >
\end{aligned}
$$

The last step is to rebuild the *RMCells*. The function Ω_{if} accomplishes this and its definition is

$$\Omega_{if} :: \textbf{set of } predicate \rightarrow (GuardedCommands, \textbf{set of } predicate)* \rightarrow RMCell*$$
$$\Omega_{if}\ ops = (if\ \ ops)*$$

where

$$if :: \textbf{set of } predicate \rightarrow (GuardedCommands, \textbf{set of } predicate) \rightarrow RMCell$$
$$if\ \ ops\ (gc, nps) = (\textbf{if}\ \ gc\,\textbf{fi}, ops \cup nps)$$

In our example we have

$$\begin{aligned}
&\Omega_{if}\ \{x \geq 10\} \\
&\quad < (x \geq 0 \rightarrow x : [x = x_0 + 1] [\!]\ x < 0 \rightarrow x := x - 1, \\
&\qquad \{x \geq 0 \Rightarrow \text{true}, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}), \\
&\quad\ (x \geq 0 \rightarrow x = x + 1 [\!]\ x < 0 \rightarrow x := x - 1, \\
&\qquad \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) > \\
&=< (\textbf{if}\ \ x \geq 0 \rightarrow x : [x = x_0 + 1] [\!]\ x < 0 \rightarrow x := x - 1\ \ \textbf{fi}, \\
&\quad \{x \geq 0 \Rightarrow \text{true}, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}), \\
&\quad (\textbf{if}\ \ x \geq 0 \rightarrow x = x + 1 [\!]\ x < 0 \rightarrow x := x - 1\ \ \textbf{fi}, \\
&\quad \{x = x_0 \wedge x > 0 \Rightarrow x + 1 = x + 0 + 1, x = x_0 \wedge x < 0 \Rightarrow x - 1 = x_0 - 1\}) >
\end{aligned}$$

We have then the definition of the structural combinator $\boxed{\textbf{if}}\_\boxed{\textbf{fi}}$

$$\begin{aligned}
&([\![\boxed{\textbf{if}}\,tactics\boxed{\textbf{fi}}]\!]\Gamma_L\ \Gamma_T)(\textbf{if}\ \ GC\ \ \textbf{fi}, proofs) = \\
&\quad \Omega_{if}\ (proofs)\ (mkGC\ (brG\ \ GC)\ (\textstyle\prod(apply\ (([\![_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))
\end{aligned}$$

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

$$\begin{aligned}
&([\![\boxed{\textbf{do}}\,tactics\boxed{\textbf{od}}]\!]\Gamma_L\ \Gamma_T)(\textbf{do}\ \ GC\ \ \textbf{od}, proofs) = \\
&\quad \Omega_{do}\ (proofs)\ (mkGC\ (brG\ \ GC)\ (\textstyle\prod(apply\ (([\![_]\!]\Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))
\end{aligned}$$

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

$$\Omega_{do} :: \textbf{set of }\ predicate \rightarrow (GuardedCommandsl, \textbf{set of }\ predicate)* \rightarrow RMCell*$$
$$\Omega_{do}\ ops = (do\ \ ops)*$$

where

$$do :: \textbf{set of }\ predicate \rightarrow (GuardedCommands, \textbf{set of }\ predicate) \rightarrow RMCell$$
$$do\ \ ops\ (gc, nps) = (\textbf{do}\ \ gc\,\textbf{od}, ops \cup nps)$$

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

$$\begin{aligned}
&([\![\boxed{\textbf{var}}\ t\boxed{]\!]}]\!]\Gamma_L\ \Gamma_T)\ ([\![\textbf{var}\ \ d \bullet p]\!], proofs) = \\
&\quad (var\ \ d) *\ ([\![t]\!]\Gamma_L\ \Gamma_T\ (p, proofs))
\end{aligned}$$

where

$$var :: declaration \rightarrow RMCell \rightarrow RMCell$$
$$var\ d\ (p, proofs) = (\llbracket \mathbf{var}\ \ d \bullet p \rrbracket, proofs)$$

This function takes two arguments. The first argument is a variable declaration and the second is an *RMCell*. It returns an *RMCell* which has as its program a variable block with the variable declaration given as argument, and the program of the *RMCell* as the program of this variable block. The set of proof obligations of the resulting *RMCell* is the set of proof obligations of the *RMCell* given as argument.

For the structural combinator $\boxed{\mathbf{cons}}\_\boxed{\rrbracket}$ we have the following definition

$$(\llbracket \boxed{\mathbf{cons}}\ t \boxed{\rrbracket} \rrbracket \Gamma_L\ \Gamma_T)\ (\llbracket \mathbf{cons}\ \ d \bullet p \rrbracket, proofs) =$$
$$(cons\ \ d) * \ (\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ (p, proofs))$$

where

$$cons :: declaration \rightarrow RMCell \rightarrow RMCell$$
$$cons\ d\ (p, proofs) = (\llbracket \mathbf{cons}\ \ d \bullet p \rrbracket, proofs)$$

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

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

$$(\llbracket \boxed{\mathbf{pmain}}\ \ t \boxed{\rrbracket} \rrbracket \Gamma_L\ \Gamma_T)\ (\llbracket \mathbf{proc}\ \ pname = body \bullet p \rrbracket, proofs) =$$
$$(procm\ pname\ body) * \ (\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ (p, proofs))$$

where

$$name \rightarrow program \rightarrow RMCell \rightarrow RMCell$$
$$procm\ pname\ body\ (p, proofs) = (\llbracket \mathbf{proc}\ \ pname = body \bullet p \rrbracket, proofs)$$

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

$$(\llbracket \boxed{\mathbf{pmainvariant}}\ \ t \boxed{\rrbracket} \rrbracket \Gamma_L\ \Gamma_T)$$
$$(\llbracket \mathbf{proc}\ \ pname = body\, \mathbf{variant}\ \ v\, \mathbf{is}\ \ e \bullet p \rrbracket, proofs) =$$
$$(variant\ pname\ body\ v\ e) * \ (\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ (p, proofs))$$

where

$$variant :: name \rightarrow program \rightarrow variable \rightarrow expression \rightarrow RMCell \rightarrow RMCell$$
$$variant\ pname\ body\ v\ e\ (p, proofs) =$$
$$(\llbracket \mathbf{proc}\ \ pname = body\ \ \mathbf{variant}\ \ v\ \ \mathbf{is}\ \ e \bullet p \rrbracket, proofs)$$

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

$$(\llbracket \boxed{\textbf{pbody}}\ t \boxed{\rrbracket}\ \rrbracket \Gamma_L\ \Gamma_T)\ (\llbracket \textbf{proc}\ \ pname = body \bullet p \rrbracket, proofs) =$$
$$(procb\ \ pname\ \ p) *\ \ (\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ (body, proofs))$$

where

$$procb :: name \rightarrow program \rightarrow RMCell \rightarrow RMCell$$
$$procb\ pname\ \ main\ (body, proofs) = (\llbracket \textbf{proc}\ \ pname = body \bullet main \rrbracket, proofs)$$

The structural combinator $\boxed{\textbf{pbodyvariant}}\ \ \_\boxed{\rrbracket}$ is used to apply a tactic to the procedure body of a variant block. Its definition is shown below.

$$(\llbracket \boxed{\textbf{pbodyvariant}}\ \ t \boxed{\rrbracket}\ \rrbracket \Gamma_L\ \Gamma_T)$$
$$(\llbracket \textbf{proc}\ \ pname = body\ \textbf{variant}\ \ v\ \textbf{is}\ \ e \bullet p \rrbracket, proofs) =$$
$$(variantb\ \ pname\ \ v\ \ e\ \ p) *\ \ (\llbracket t \rrbracket \Gamma_L\ \Gamma_T\ (body, proofs))$$

where

$$variantb :: name \rightarrow variable \rightarrow expression \rightarrow program \rightarrow RMCell \rightarrow RMCell$$
$$variantb\ \ pname\ \ v\ \ e\ \ main\ (body, proofs) =$$
$$(\llbracket \textbf{proc}\ \ pname = body\ \ \textbf{variant}\ \ v\ \ \textbf{is}\ \ e \bullet main \rrbracket, proofs)$$

The structural combinator $\boxed{\textbf{val}}\ \ \_$ is used when we have an argument passed by value in the program of an *RMCell*. Its definition is

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

where

$$val :: variable \rightarrow type \rightarrow variable* \rightarrow RMCell \rightarrow RMCell$$
$$val\ \ v\ \ T\ \ pars\ (p, proofs) = ((\textbf{val}\ v : T \bullet p)(pars), proofs)$$

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

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

where

$$res :: variable \rightarrow type \rightarrow variable* \rightarrow \rightarrow RMCell$$
$$res\ \ v\ \ T\ \ pars\ (p, proofs) = ((\textbf{res}\ v : T \bullet p)(pars), proofs)$$

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

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

$$valres :: variable \rightarrow type \rightarrow variable* \rightarrow\rightarrow RMCell$$
$$valres\ \ v\ \ T\ \ pars\ (p, proofs) = ((\mathbf{val\_res}\ v : t \bullet p)(pars), proofs)$$

The semantics of **applies to** *program* **do** *tactic* needs two important constructs in its definition. First we define the function *equals* which yields a tactic that succeeds only if it is presented with an *RMCell* matching the argument:

$$equals :: RMCell \rightarrow Tactic$$
$$equals\ \ r\ \ r = [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ \ r$$
$$equals\ \ r\ \ h = [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ \ r\ \ where\ \ \ h \neq r$$

This function checks if two *RMCells* match. The second construct is the tactic

$$\mathbf{con}\ \ v \bullet\ \ t$$

which introduces v as a set of free variables chosen from an appropriate syntatic class (denoted *TERM* below) in t, angelically chosen so that as many choices as possible succeed. Its semantics is

$$[\![\mathbf{con}\ \ v \bullet\ \ t(v)]\!]\ \Gamma_L\ \Gamma_T = [\![|_{v \in TERM} t(v)]\!]\ \Gamma_L\ \Gamma_T$$

where $|_{v \in TERM} t(v)$ is an alternation of all tactics which can be obtained from t changing v by all possible values that v can have. This is an infinite alternation. Formally, an infinite alternation can be defined as

$$|_{i=0}^{\infty}\ \mu\, X \bullet F(X_0)$$
$$where\ \ \ F(X_i) = f(i)\ \big|\ F(X_{i+1})$$

Now we may define

$$[\![\mathbf{applies\ to}$$
$$p$$
$$\mathbf{do}$$
$$tacticBody]\!]\ \Gamma_L\ \Gamma_T)\ (p_1, proofs)\ \ =$$
$$[\![\mathbf{con}\ \ p \bullet equals\ (p, proofs);\ \ tacticBody(args)]\!]\ \Gamma_L\ \Gamma_T\ \ (p_1, proofs)$$

This tactic makes an alternation of all possible formats of p with its meta-variables. If p_1 can be regarded as a program that matches the format of p then the tactic *tacticBody* is applied to p_1.

4.3 Formal Definition of Tactic Declarations and Programs

We have then the definition of the semantic function for *tacticDecl* and *tacticProg*. First, the *tacticDecl* overwrites the tactics environment as follows

$$
\begin{array}{l}
[\![\_]\!] \_ \_ :: tacticDecl \rightarrow LawsEnvironment \rightarrow TacticsEnvironment \rightarrow TacticsEnvironment\\
[\![\mathbf{Tactic}\ tname\ (args)\\
\quad tactic\\
\mathbf{proof\ obligations}\\
\quad \mathbf{set\ of}\ \ predicate\\
\mathbf{program\ generated}\\
\quad program\\
\mathbf{end}]\!]\ \Gamma_L\ \Gamma_T =\\
\Gamma_T \oplus \{tname \mapsto \{v \in TERM \bullet v \rightarrow [\![tactic[args \setminus v]]\!]\Gamma_L\ \Gamma_T)\}\}
\end{array}
$$

where $\oplus$ is the overriding operator. The presence of the clauses **proof obligations** and **program generated** does not change the semantics of a *tacticDecl*. The idea is to overwrite the tactics environment to include in the domain the tactic name *tname*. This element is mapped to a new function which, for each possible argument $v \in TERM$, v is mapped to the tactic semantic, replacing the arguments of the tactic for v.

Then, a *tacticProg*, which is a sequence of *tacticDecl* is defined as

$$
\begin{array}{l}
[\![\_]\!] \_ \_ :: tacticProg \rightarrow LawsEnvironment \rightarrow TacticsEnvironment \rightarrow TacticsEnvironment\\
[\![tacticDecls\ tactic]\!]\ \Gamma_L\ \Gamma_T =\\
[\![tactic]\!]\ \Gamma_L\ (decl\ tacticDecls\ \Gamma_L\ \Gamma_T)
\end{array}
$$

where the auxiliar function *decl* is defined below

$$
\begin{array}{l}
decl\ \ \varnothing\ \ \Gamma_L\ \Gamma_T = \Gamma_T\\
decl\ (tacticDecl_1\ \ tacticDecls)\ \Gamma_L\ \Gamma_T =\\
\qquad (([\![tacticDecl_1]\!]\ \Gamma_L\ \Gamma_T) \oplus (decl\ tacticDecls\ \Gamma_L\ \Gamma_T)
\end{array}
$$

where $\varnothing$ is the empty declaration.

The idea is to sequentially overwrite the tactics environment. If we have just one tactic declaration then the tactics environment is overwritten just once. Otherwise it is overwritten for this tactic declaration, this result tactics environment is overwritten for the next tactic declaration, and so on.

5 Tacticals

Most tactics languages include a definition of a $repeat(n, t)$ tactical which is defined as follows:

$$
\begin{array}{l}
repeat(0, t) = \mathbf{skip}\\
repeat(n + 1, t) = t;\ repeat(n, t)
\end{array}
$$

$repeat(n, t)$ applies the tactic t the number of times specified by n.

Another tactical makes a robust application of a tactic t. This is called *try* and is defined as

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

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

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

This tactic applies t as many times as possible, terminating (with success) when t fails to apply.

Two tacticals are used to make tactic assertions. The tactic *succs* t fails whenever t fails, and behaves like **skip** whenever t succeeds. On other hand, *fails* t behaves like **skip** if t fails, and fails if t succeeds.

$$fails\ \ t\ \ r = (\mathbf{skip} \lhd\ \ t\ \ r = <> \rhd \mathbf{fail})\ r$$
$$succs\ \ t\ \ r = (\mathbf{fail} \lhd\ \ t\ \ r = <> \rhd \mathbf{skip})\ r$$

They are defined using the infix conditional notation $t_1 \lhd p \rhd t_2$ which can be read as **if** p **then** t_1 **else** t_2.

6 Simple Laws

In this section we present laws proposed in the literature for Angel. In Appendix C we prove their soundness in the context of the RTL semantics. In Section 7 we show how these laws can be used to reduce tactics written in a subset of RTL to a normal form.

6.1 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) $\mathbf{skip}\ ;\ t = t$
Law 1.(b) $t = \mathbf{skip};\ t$
Law 2.(a) $t \mid \mathbf{fail} = t$
Law 2.(b) $t = \mathbf{fail} \mid t$
Law 3.(a) $t;\ \mathbf{fail} = \mathbf{fail}$
Law 3.(b) $\mathbf{fail} = t;\ \mathbf{fail}$

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.

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.

6.2 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 laws and tactics are unchanged by applications of *cut*.

Law 8. $!\,\mathbf{skip} = \mathbf{skip}$
Law 9. $!\,\mathbf{fail} = \mathbf{fail}$
Law 10. $!(\mathbf{law}\ \ l\ (args)) = \mathbf{law}\ \ l\ (args)$

The use of the cur operator allows the sequential composition to distribute over alternation on the left.

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. $!(t_1 \mid t_2 \mid t_1) =!(t_1 \mid t_2)$

The **skip** becomes a left-zero for alternation, alternation becomes idempotent, and the *cut* itself is also idempotent.

Law 17. $!(\mathbf{skip} \mid t) = \mathbf{skip}$
Law 18. $!(t \mid t) =!t$
Law 19. $!!t =!t$

6.3 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.

Law 20. $succs\ t;\ t = t$
Law 21. $fails\ t;\ t = \mathbf{fail}$
Law 22. $fails\ t = fails\ !t = fails\ !t$
Law 23. $fails(succs\ t) = fails\ t$
Law 24. $fails\ t_1;\ fails\ t_2 = fails\ t_2;\ fails\ t_1$

Law 25. $!(t_1 \mid t_2) = !t_1 \mid (fails\ t_1;\ !t_2)$
Law 26. $!(t_1;\ t_2) = !(t_1;\ succs\ t_2);\ !t_2$
Law 27. $succs(t_1 \mid t_2) = !(succs\ t_1 \mid succs\ t_2)$
Law 28. $fails(t_1 \mid t_2) = fails\ t_1;\ fails\ t_2$
Law 29. $succs\ s;\ succs(s;\ t) = succs(s;\ t)$
Law 30. $fails\ s = fails\ s;\ fails(s;\ t)$
Law 31. $!s;\ fails\ t = fails(!s;\ t);\ !s$
Law 32. $fails(fails(s;\ t)) = succs\ s \mid (fails\ s;\ fails\ t)$
Law 33. $fails(s;\ fails\ t) = fails\ s \mid succs(s;\ t)$
Law 34. $fails\ s;\ succs(s;\ t) = \mathbf{fail}$
Law 35. $fails(t;\ d) = fails(t;\ succs\ d)$
Law 36. $!s;\ succs\ t = succs\ (!s;\ t);\ !s$
Law 37. $fails(fails\ t) = succs\ t$
Law 38. $fails\ t_1;\ succs\ t_2 = succs\ t_2;\ fails\ t_1$
Law 39. $succs\ t_1;\ succs\ t_2 = succs\ t_2;\ succs\ t_1$
Law 40. $succs(succs\ t) = succs\ t$
Law 41. $succs(fails\ t) = fails\ t$
Law 42. $succs(t;\ d) = succs(t;\ succs\ d)$
Law 43. $succs\ t = succs\ !t = !succs\ t$
Law 44. $succs\ \mathbf{skip} = \mathbf{skip}$
Law 45. $succs\ \mathbf{fail} = \mathbf{fail}$
Law 46. $fails\ \mathbf{skip} = \mathbf{fail}$
Law 47. $fails\ \mathbf{fail} = \mathbf{skip}$
Law 48. $fails\ t;\ fails\ t = fails\ t$
Law 49. $succs\ t;\ succs\ t = succs\ t$
Law 50. $succs\ t;\ fails\ t = fails\ t;\ succs\ t = \mathbf{fail}$
Law 51. $fails\ t \mid succs\ t = succs\ t \mid fails\ t = \mathbf{skip}$
Law 52. $succs(t \mid u) = succs\ t \mid (failst;\ succs\ u)$
Law 53. $succs(fails\ s;\ t)) = fails\ s;\ succs\ t$
Law 54. $succs(succs\ s;\ t) = succs\ s;\ succs\ t$
Law 55. $succs(s;\ fails\ t) = succs\ s;\ fails(s;\ t)$
Law 56. $succs(s;\ succs\ t) = succs\ s;\ succs\ (s;\ t)$
Law 57. $fails\ (succs\ s;\ t) = fails\ s \mid (succs\ s;\ fails\ t)$
Law 58. $fails(s;\ succs\ t) = fails\ s \mid (succs\ s;\ fails\ (s;\ t))$

6.4 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. $!(t_1\boxed{;}t_2) = !t_1\boxed{;}!t_2)$

6.5 Laws on *equals*

Some laws on *equals* are shown in this section.

Law 63. $equals\ r_1;\ equals\ r_2 = fail\ if\ r_1 \neq r_2$
Law 64. $equals\ r;\ equals\ r = equals\ r$
Law 65. $!equals\ r = equals\ r$

6.6 Laws on 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.

Law 66. $(\mathbf{con}\ \ v \bullet t_1;\ t_2) = (\mathbf{con}\ \ v \bullet t_1);\ t_2$ provided $v \notin \phi t_2$
Law 67. $(\mathbf{con}\ \ v \bullet t_1;\ t_2) = t_1;\ (\mathbf{con}\ \ v \bullet t_2)$ provided $v \notin \phi t_1$ and $!t_1 = t_1$
Law 68. $(\mathbf{con}\ \ v \bullet t) = t$ provided $v \notin \phi t$
Law 69. $(\mathbf{con}\ \ v \bullet t) = (\mathbf{con}\ \ u \bullet t[v \setminus u])$ provided $u \notin \phi t$
Law 70. $equals\ r;\ (\mathbf{con}\ \ r' \bullet equals\ r';\ t) = equals\ r;\ t[r' \setminus r])$

7 Normal Form

In the following subsections we prove that tactics written in a subset of RTL have a normal form and that it is unique. First, we present the cut-free normal form. Then we present the pre-normal form, and finally we present the general normal form.

The idea behind the proof is to define a normal form for tactics, to show that every tactic can be transformed into a unique normal form using the laws in the previous section, and finally to show that if two tactics are equivalent then they have the same normal form.

In the following proofs we shall use the notation

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

to represent the alternate composition $t_{i_1} \mid ... \mid t_{i_n}$, 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 will denote **fail**. In the same way we have the notation

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

with the difference that the empty sequential composition denotates **skip**.

7.1 Cut-free Normal Form

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

$$|_{i:I} \, (;\ _{j:J_i} \, \mathbf{law} \ \ l_j)$$

where l_j are names of laws in the domain of $lawsenvironment$.

The idea of this normal form is that all tactics, even the more elaborated, written in the subset of RTL considered below, can be expressed as an alternation of sequences of laws applications.

Lemma 1. Any tactic expressed using basic laws, alternation, sequential composition, **skip**, and **fail** can be transformed into a unique tactic in cut-free normal form using the laws in the previous section.

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

Base cases :
$\mathbf{skip} = (;\ _{\varnothing})$

$\mathbf{fail} = |_{\varnothing}$

$\mathbf{law} \ \ l = |_{i:I} \, (\mathbf{law} \ \ l)$

Inductive step : We assume that
$t_1 = (|_{i:I} \, (;\ _{j:J_i} \, \mathbf{law} \ \ l_j))$ and
$t_2 = (|_{k:K} \, (;\ _{l:L_k} \, \mathbf{law} \ \ l_l))$

We must prove that

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

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

$$(|_{i:I} \, (;\ _{j:J_i} \, \mathbf{law} \ \ l_j)) \mid (|_{k:K} \, (;\ _{l:L_k} \, \mathbf{law} \ \ l_l))$$

to the cut-free normal form.

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

$$(|_{j:J} \, (;\ _{i:I_j} \, \mathbf{law} \ \ l_i));\ (|_{k:K} \, (;\ _{l:L_k} \, \mathbf{law} \ \ l_l))$$

First we generalize the Laws 6 and 7 to
Law 6*. $(|_{i:I} \, t_i);\ t_3 = (|_{i:I} \, t_i;\ t_3)$
Law 7*. $t_1;\ (|_{i:I} \, t_i) = (|_{i:I} \, t_1;\ t_i)$

The proofs of these generalized laws are consequence of the Laws 6 and 7 and are not shown in this document.

And we proceed with the proof

$$
\begin{array}{ll}
(\big|_{j:J}\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (\big|_{k:K}\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)) & \\
= \big|_{j:J}\ ((;\ _{i:I_j}\ \mathbf{law}\ \ l_i);\ (\big|_{k:K}\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l))) & [\text{Law } 6^*] \\
= \big|_{j:J}\ (\big|_{k:K}\ ((;\ _{i:I_j}\ \mathbf{law}\ \ l_i);\ ((;\ _{l:L_k}\ \mathbf{law}\ \ l_l))) & [\text{Law } 7^*]
\end{array}
$$

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

□

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

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

Then, we prove that if two tactics in normal form are equivalent, then they are sintatically equal.

Lemma 2. If two tactics in normal form are equivalent then they are identical.

Proof. This is proved showing that there is no point where the tactics differ. We also use the following notation: Let the laws which are used in the tactics be l_1, l_2, ..., and assume that the *RMCells* are indexed by sequences of laws numbers, so that $\mathbf{law}\ \ l_n\ r_{<>} = r_{<n>}$ and $\mathbf{law}\ \ l_n\ r_l = r_{l^\frown <n>}$. For example we have that

$$((\mathbf{law}\ \ l_1;\ \mathbf{law}\ \ l_2;\ \mathbf{law}\ \ l_3) \,\big|\, \mathbf{law}\ \ l_4)\ r_{<>} \qquad =< r_{<1,2,3>}, r_{<4>} >$$

Using this law instantiation a tactic applied to $r_{<>}$ will produce an account of the tactic's normal form (with the alternation of the sequential composition of the laws in the respective *RMCells* in the result list) and so the result is immediate.

□

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 $t_2 \equiv t_{2n}$. 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, $t_1 \equiv t_2$.

□

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 given above are complete for the cut-free finite (non-recursive) RTL.

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

□

We have presented in this subsection the cut-free normal form. However, the *cut* operator gives more scope to structuring tactics, and so, in the next sections we are concerned in extending the subset of RTL considered, so that it also contains the *cut* operator.

7.2 Pre-Normal Form

Now, consider a subset of RTL that contains also the *cut*(!) operator and the tactic assertions *succs* and *fails*.

We first extend the earlier definition of the sequential tactics, so that it also contains these operators. We use it in the Lemma 3 below.

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 tactic t, 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 will be very usefull.

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

Proof. By structural induction, using definition 3 above.

Base cases :
$\textbf{skip} =! \textbf{skip}$ [Law 8]

$\textbf{fail} =! \textbf{fail}$ [Law 9]

$\textbf{law}\ \ l =!(\textbf{law}\ \ l)$ [Law 10]

Inductive step : We assume that t is sequential

We must consider the following cases

a) $!t$

$= !(t \mid \mathbf{fail})$ [Law 2]

$= !(!t \mid \mathbf{fail})$ [Law 14]

$= !(!t)$ [Law 2]

b) $\mathit{fails}\ t = !\mathit{fails}\ t$ [Law 22]

c) $\mathit{succs}\ t = !\mathit{succs}\ t$ [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]

$\square$

Now it is possible to prove a completeness result for the finite (non-recursive) RTL, including the cut operator, but excluding the structural combinators and the **applies to _ do_** operator. The proof will be 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

$$|_{j:J}\ g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i)$$

where g_j are guards of the form $\mathit{succs}(;\ _k\ l_k)$ or $\mathit{fails}(;\ _k\ l_k)$ or a possible empty sequential composition of such guards, and l_i are instances of laws.

Now, we want to prove that all the tactics written in the subset of RTL 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:

$\mathbf{skip} = \mathit{succs}\ \ \mathbf{skip};\ (;\ _{i:\varnothing})$

$\mathbf{fail} = |_{j:\varnothing}$

$\mathbf{law}\ \ l = succs\ (\mathbf{law}\ \ l);\ \mathbf{law}\ \ l$ [Law 20]

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

We must prove that

a) $t_1 \mid t_2$ can be put in pre-normal form

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

$= (|_{j:J}\ g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i)) \mid (|_{k:K}\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)))$

to the cut-free normal form.

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

$= (|_{j:J}\ g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (|_{k:K}\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l))$
$= |_{j:J}\ ((g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (|_{k:K}\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)))$ [Law 6*]
$= |_{j:J}\ (!(g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (|_{k:K}\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)))$ [Lemma 3]

We can generalise the Law 11 as

Law 11*. $!t_1;\ (|_{i:I}\ t_i) = |_{i:i}\ t_1;\ t_i$

The proof of this generalized law is consequence of the Law 11 and is not shown in this document.

And so we have

$|_{j:J}\ (!(g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (|_{k:K}\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)))$ [Lemma 3]
$= |_{j:J}\ (|_{k:K}\ (!(g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i));\ (g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l))))$ [Law 11*]
$= |_{j:J}\ (|_{k:K}\ (g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i);\ g_k;\ (;\ _{l:L_k}\ \mathbf{law}\ \ l_l)))$ [Lemma 3]

This tactic has the form

$|_{i:I}\ (g_{1_i};\ t_{1_i};\ g_{2_i};\ t_{2_i})$

Since t_{1_i} are sequences of laws 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

$$
\begin{array}{ll}
succs\ (\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1;\ succs\ (\textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3);\ \textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3 & \\
\ \mid succs\ (\textbf{law}\ \ l_4);\ \textbf{law}\ \ l_4;\ succs\ (\textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6);\ \textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6 & \\
= succs\ (\textbf{law}\ \ l_1);\ !\,\textbf{law}\ \ l_1;\ succs\ (\textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3);\ !(\textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3) & \\
\ \mid succs\ (\textbf{law}\ \ l_4);\ !\,\textbf{law}\ \ l_4;\ succs\ (\textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6);\ !(\textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6) & \text{[Lemma 3]} \\
= succs\ (\textbf{law}\ \ l_1);\ succs(!\,\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3);\ !\,\textbf{law}\ \ l_1;\ !(\textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3) & \\
\ \mid succs\ (\textbf{law}\ \ l_4);\ succs(!\,\textbf{law}\ \ l_4;\ \textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6);\ !\,\textbf{law}\ \ l_4;\ !(\textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6) & \text{[Law 36]} \\
= succs\ (\textbf{law}\ \ l_1);\ succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2;\ \textbf{law}\ \ l_3 & \\
\ \mid succs\ (\textbf{law}\ \ l_4);\ succs(\textbf{law}\ \ l_4;\ \textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6);\ \textbf{law}\ \ l_4;\ \textbf{law}\ \ l_5;\ \textbf{law}\ \ l_6 & \text{[Lemma 3]}
\end{array}
$$

The resulting tactic will be in pre-normal form.

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

The tactic $!t$ can be normalised 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

$$
\begin{array}{ll}
!(\mid_{j:J}\ g_j;\ (;\ _{i:I_j}\ \textbf{law}\ \ l_i)) & \\
= !(g_{1_j};\ (;\ _{i:I_{1_j}}\ \textbf{law}\ \ l_i))\ \mid & \\
\quad fails(g_{1_j};\ (;\ _{i:I_{1_j}}\ \textbf{law}\ \ l_i));\ !(\mid_{j:J\setminus\{1\}}\ g_j;\ (;\ _{i:I_j}\ \textbf{law}\ \ l_i)) & \text{[Law 25]} \\
= (g_{1_j};\ (;\ _{i:I_{1_j}}\ \textbf{law}\ \ l_i))\ \mid & \\
\quad fails(g_{1_j};\ (;\ _{i:I_{1_j}}\ \textbf{law}\ \ l_i));\ !(\mid_{j:J\setminus\{1\}}\ g_j;\ (;\ _{i:I_j}\ \textbf{law}\ \ l_i)) & \text{[Lemma 3]}
\end{array}
$$

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

$$
\begin{array}{ll}
!(succs(\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1 & \\
\quad \mid succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2) & \\
= !(succs(\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1) & \\
\quad \mid (fails(succs(\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1);\ !(succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2)) & \text{[Law 25]} \\
= (succs(\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1) & \\
\quad \mid (fails\ (\textbf{law}\ \ l_1)\mid succs(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_1));\ (succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2)) & \\
 & \text{[Lemma 3]} \\
= (succs(\textbf{law}\ \ l_1);\ \textbf{law}\ \ l_1) & \\
\quad \mid (fails\ (\textbf{law}\ \ l_1);\ succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2) & \\
\quad \mid (succs(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_1);\ succs(\textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2);\ \textbf{law}\ \ l_1;\ \textbf{law}\ \ l_2) & \text{[Law 6]}
\end{array}
$$

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

Generalizing Law 28 we have that

Law 28*. $fails(|_{i:I}\ t_i) =;\ _{i:I} fails(t_i)$

The proof of this generalized law is consequence of the Law 27 and is not shown in this document.

and so

$$\begin{array}{ll} fails(|_{j:J}\ g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i)) & \\ =\ ;\ _{j:J} fails(g_j;\ (;\ _{i:I_j}\ \mathbf{law}\ \ l_i)) & [\text{Law } 28^*] \end{array}$$

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 so 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.

$\square$

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

7.3 General Normal Form

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

$s;\ t$
$(succs\ s);\ s;\ t$
$(succs\ s;\ t);\ s;\ t$
$(succs\ s);\ (succs\ s;\ t);\ s;\ t$

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 have some concepts in mind. Law sequences are sequences of laws applications in the form $\mathbf{law}\ \ l_1;\ ...;\ \mathbf{law}\ \ l_n$. A set T of such sequences of laws is prefixed closed if for any law sequence t in T, T also contains all initial subsequences of t. For example, the set

$$T = \{\mathbf{law}\ \ l_1, \mathbf{law}\ \ l_1;\ \mathbf{law}\ \ l_2, \mathbf{law}\ \ l_1;\ \mathbf{law}\ \ l_2;\ \mathbf{law}\ \ l_3\}$$

is prefixed closed, but the set

$$T = \{\mathbf{law}\ \ l_1, \mathbf{law}\ \ l_1;\ \mathbf{law}\ \ l_2;\ \mathbf{law}\ \ l_3\}$$

is not, since the element $\mathbf{law}\ \ l_1, \mathbf{law}\ \ l_2$ 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 sequence T, if it has the form

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

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 sequence t, g_j contains *succs* t, it must not contain *fails* s, for s any prefix 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 that of all the alternate clauses in v_j, i.e., if $v_j = |_k\ v_{j_k}$, 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 sequence t for which g_j contains *succs* t and g_j contains *fails* t (or vice versa).

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

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

$g_i \neq$ **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**.

succs t; $g_j = g_j$ or *fails* t; $g_j = g_j$: Again the maximality property guarantees that all sequence of laws 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.

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.

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 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

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

if we compose these two guards, we have

$$\begin{array}{ll}
succs(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_2);\ fails(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_2) & \\
= succs(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_2);\ fails(\textbf{law}\ \ l_2) & [\text{Law } 24] \\
= succs(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_1);\ fails(\textbf{law}\ \ l_2) & [\text{Law } 48] \\
= \textbf{fail};\ fails(\textbf{law}\ \ l_2) & [succs\ t = \textbf{skip} \Rightarrow fails\ t = \textbf{fail}] \\
 & [succs\ t = \textbf{fail} \Rightarrow fails\ t = \textbf{skip}] \\
= \textbf{fail} & [\text{Law } 3]
\end{array}$$

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

$$\begin{array}{l}
\ldots \\
\mid\ \ldots;\ succs\ t;\ \ldots;\ v_n \\
\ldots \\
\mid\ \ldots;\ fails\ t;\ \ldots;\ v_m \\
\ldots
\end{array}$$

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

$$\begin{array}{l}
\ldots \\
\mid succs\ t;\ \ldots;\ v_n \\
\ldots \\
\mid fails\ t;\ \ldots;\ v_m \\
\ldots
\end{array}$$

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

$$\begin{array}{l}
\ldots \\
\mid succs\ t;\ \ldots;\ v_n \\
\mid fails\ t;\ \ldots;\ v_m \\
\ldots
\end{array}$$

Therefore, we have for an arbitrary pair of adjacent alternation branches of a tactic in general normal form can be considered to have the form $succs\ t;\ s_1 \mid fails\ t;\ s_2$. In the last step we have re-ordered the outermost alternation swapping branches that do not start with an assertion (*succs* or *fails*) about a same tactic t. Now we want to re-order branches, like those above.

These pairs can also be reordered as seen below:

$succs\ t;\ s_1 \mid fails\ t;\ s_2$
$= (fails\ t \mid succs\ t);\ (succs\ t;\ s_1 \mid fails\ t;\ s_2)$ [Laws 51 and 1]
$= (fails\ t;\ succs\ t;\ s_1)$
$\mid (fails\ t;\ fails\ t;\ s_2)$
$\mid (succs\ t;\ succs\ t;\ s_1)$
$\mid (succs\ t;\ fails\ t;\ s_2)$ [Laws 6 and 11,Lemma 3]
$= fails\ t;\ s_2 \mid succs\ t;\ s_1$ [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 will be 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 sintactically different, but equivalent, using the commutative laws on *succs* and *fails*, and so, the tactics are equivalent.

Lemma 5. Given a sufficiently large prefix-closed set of law sequences, T, any tactic in pre-normal form can be put into a unique normal form, relative to T (unique modulo reordering of the guards), using laws drawn from the set given above. Sufficiently large means that it must at least contain the minimal set of law sequences determined by consideration of the tactic t in pre-normal form: that set must contain all the instances of law sequences $(;\ _{i:I_j}\ \mathbf{law}\ \ l_i)$.

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 sequences in T:

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

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

$$\big|_{i:I}\ \ g_i$$

where $g_i = succs\ t$ or $g_i = fails\ t$ 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 ilustrate these first steps let us have $T = \{< l_1 >, < l_1, l_2 >, < l_1, l_2, l_3 >\}$. We have the tactic formed from all possible guards for law sequences in T as seen below:

$$
\begin{array}{l}
(succs\ l_1 \mid fails\ l_1); \\
(succs\ l_1;\ l_2 \mid fails\ l_1;\ l_2); \\
(succs\ l_1;\ l_2;\ l_3 \mid fails\ l_1;\ l_2;\ l_3)
\end{array}
$$

Using the distributive laws (applying Lemma 3) we have

$$
\begin{array}{ll}
succs\ l_1;\ succs\ (l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3) & [1] \\
\mid succs\ l_1;\ fails\ (l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3) & [2] \\
\mid fails\ l_1;\ succs\ (l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3) & [3] \\
\mid fails\ l_1;\ fails\ (l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3) & [4] \\
\mid succs\ l_1;\ succs\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [5] \\
\mid succs\ l_1;\ fails\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [6] \\
\mid fails\ l_1;\ succs\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [7] \\
\mid fails\ l_1;\ fails\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [8]
\end{array}
$$

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

$$
\begin{array}{ll}
succs\ l_1;\ succs\ (l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3) & [1] \\
\mid succs\ l_1;\ succs\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [5] \\
\mid succs\ l_1;\ fails\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [6] \\
\mid fails\ l_1;\ fails\ (l_1;\ l_2);\ fails(l_1;\ l_2;\ l_3) & [8]
\end{array}
$$

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. They also have 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

$$(|_{i:I'}\ g_i);\ (|_{j:J}\ h_j;\ (;\ _{k:K_j} r_k))$$

Applying the distributive laws we have

$$|_{i:I'}\ |_{j:J}\ g_i;\ h_j;\ (;\ _{k:K_j} r_k)$$

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, $g_i;\ h_i = g_i$, or that g_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 $g_i;\ h_i =$ **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**.

$$|_{i:I'} \; |_{j:J'_i} \; g_i; \; (; \; _{k:K_j} r_k)$$

The next step is to omit (using the commutative laws and Law 2) the inner alternation branches whose guards contains an instance of $fails(; \; _k r_k)$, 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:

$$|_{i:I'} \; (g_i; \; |_{j:J''_i} \; (; \; _{k:K_j} r_k))$$

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.

□

Lemma 6. Two tactics in general normal form (relative to some sufficiently large prefixed-closed set of rule 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 we refer to are $l_1, l_2, ..., l_n$ and the *RMCells* are decorated with pairs. The first element of this pair is a prefixed closed set of law sequences. These sequences are those which can succeed when applied to that *RMCell*. The second element is the law sequence which contains the laws which have been already successfully applied.

This is, if

$$fst(r_{s,t}) \in \operatorname{dom} \Gamma_L \; l_i \; args \Leftrightarrow \exists \, xs : s \mid head \; xs = l_i.$$

then

$$l_i \; r_{s,t} = r_{s',t \frown <r_i>} \quad where \; s' = xs \mid r_i : xs \in s$$

otherwise the laws application fails.

Analysing the behaviour of a tactic applied to different *RMCells*, we can determine the guard/body (in cut-free normal form) pairs of its normal form. The application of a tactic to an *RMCell* $r_{s,<>}$ 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 later 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 *RMCells* $r_{s,<>}$ (with s any prefixed 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

$$T = \{l_1, l_1;\ l_2, l_1;\ l_3, l_1;\ l_2;\ l_3, l_1;\ l_2;\ l_3;\ l_4\}$$

and there are only two possible initial *RMCell* which are

$$R_{<l_1, l_1;\ l_2, l_1;\ l_3, l_1;\ l_2;\ l_3, l_1;\ l_2;\ l_3;\ l_4>, <>}$$
$$R_{<l_1, l_1;\ l_2, l_1;\ l_2;\ l_3, l_1;\ l_2;\ l_3;\ l_4>, <>}$$

Now, we consider that when we apply a tactic t_1 to these two *RMCells* we get, respectively,

$$< R_{<l_3, l_3;\ l_4>, <l_1, l_2>}, R_{<l_2, l_2;\ l_3, l_2;\ l_3;\ l_4>, <l_1, l_3>} >$$
$$< R_{<l_3, l_3;\ l_4>, <l_1, l_2>} >$$

If we rebuild this tactic as described before, we have, for the first branche (from the first *RMCell*)

$$succs(l_1);\ succs(l_1;\ l_2);\ succs(l_1;\ l_3);\ succs(l_1;\ l_2;\ l_3);\ succs(l_1;\ l_2;\ l_3;\ l_4);\ (l_1;\ l_2 \mid l_1;\ l_3)$$

fails does not appear bacause $T \setminus s$ is empty for the first *RMCell*. For the second *RMCell* we have

$$succs(l_1);\ succs(l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3);\ succs(l_1;\ l_2;\ l_3;\ l_4);\ fails(l_1;\ l_3);\ (l_1;\ l_2)$$

Now, we make an alternation of these to branches, and we have the tactic in the normal form.

$$succs(l_1);\ succs(l_1;\ l_2);\ succs(l_1;\ l_3);\ succs(l_1;\ l_2;\ l_3);\ succs(l_1;\ l_2;\ l_3;\ l_4);\ (l_1;\ l_2 \mid l_1;\ l_3)$$
$$\mid$$
$$succs(l_1);\ succs(l_1;\ l_2);\ succs(l_1;\ l_2;\ l_3);\ succs(l_1;\ l_2;\ l_3;\ l_4);\ fails(l_1;\ l_3);\ (l_1;\ l_2)$$

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 *RMCell* $r_{s,<>}$ 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 guaranted by the fact that the law sequences $t_1, ..., t_n$ are those which succeeds when applied to the initial *RMCell*. So, these law 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.

□

Theorem 2. Two tactics are equivalent under all law 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 $t_2 \equiv t_{2n}$. So we can also say that $t_{1n} \equiv t_{2n}$, and then, using Lemma 2, 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 hte 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.

$\square$

We are yet to consider tactics involving recursion and the structural combinators. This is left as further work.

8 The Tactics and Examples

In this section we present some tactics and examples of refinements using these tactics. The tactics are based on the development strategies presented in [10, 2, 8].

In the design of a program involving an iteration, the main concern is to determine the iteration invariant. In [8] several strategies are presented, which we capture as tactics.

8.1 Tactic takeConjAsInv

8.1.1 The Tactic

The first strategy 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 $w : [pre, invconj \wedge notGuard]$ into an initialized iteration.

Tactic takeConjAsInv $(invBound, (lstVar, lstVal), variant)$
applies to $w : [pre, invConj \wedge notGuard]$
do **law strPost**$(invConj \wedge notGuard \wedge invBound)$;
  law seqCom$(invConj \wedge invBound)$;
  (**law assig**$(lstVar, lstVal)$ $\boxed{;}$ **law iter**$(\langle \neg\ notGuard \rangle, invConj \wedge invBound, variant)$);
proof obligations
  1.$(invConj \wedge notGuard \wedge invBound) \Rightarrow (invConj \wedge notGuard)$
  2.$pre \Rightarrow (invConj \wedge invBound)[lstVal/lstVar]$
program generated
  $lstVar := lstVal$;
  do $\neg\ notGuard \rightarrow$
    $w : [invConj \wedge invBound \wedge \neg\ notGuard,$
      $invConj \wedge invBound \wedge 0 \leq variant < variant_0]$
  od
end

This tactic has three arguments: a predicate $invBound$, a pair $(lstVar, lstVal)$, where $lstVar$ is a list of variables and $lstVal$ is a list of values, and an integer expression $variant$. This tactic applies to a specification statement $w : [pre, invConj \wedge notGuard]$ to introduce an initialized iteration whose invariant is $invConj \wedge invBound$ and whose variant is $variant$. The initialization is $lstVar := lstVal$. Typically, the predicate $invBound$ states the range limits of indexing variables of the iteration. The conjunction $invConj \wedge invBound$ 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 **seqCom**). Afterwards, the law **assig** is applied to the first program of the composition to derive the initialization, and the law **iter** to the second program in order to introduce the iteration. The first proof obligation is generated by the application of law **strPost**, and the second by the application of law **assig**.

8.1.2 Examples

$x : [n \geq 0, x^2 \leq n \wedge (x+1)^2 > n]$ $\lhd$
$\sqsubseteq$ **takeConjAsInv**$(0 \leq x, (\langle x \rangle, \langle 0 \rangle), n - x^2)$
  $\sqsubseteq$ **strPost**$(0 \leq x)$
    $x : [n \geq 0, 0 \leq x \wedge x^2 \leq n \wedge (x+1)^2 > n]$ $\lhd$

This law generates the following proof obligation

$$(0 \leq x \wedge x^2 \leq n \wedge (x+1)^2 > n) \Rightarrow (x^2 \leq n \wedge (x+1)^2 > n)$$

Most of the proof obligations generated in the examples are very simple, they follow directly from the properties of the predicate calculus and of the data type involved, mainly numbers.

$\sqsubseteq$ **seqCom**$(x^2 \leq n \wedge 0 \leq x)$
$x : [n \geq 0, x^2 \leq n \wedge 0 \leq x];$ $\lhd$
$x : [x^2 \leq n \wedge 0 \leq x, 0 \leq x \wedge x^2 \leq n \wedge (x+1)^2 > n]$ (i)
$\sqsubseteq$ **assig**$(\langle x \rangle, \langle 0 \rangle)$
$x := 0$
(i) $\sqsubseteq$ **iter**$((x+1)^2 \leq n, n - x^2)$
do $(x+1)^2 \leq n \rightarrow$
$x : [x^2 \leq n \wedge 0 \leq x \wedge (x+1)^2 \leq n, x^2 \leq n \wedge 0 \leq x \wedge 0 \leq n - x^2 < n - {x_0}^2]$ $\lhd$
od
$\sqsubseteq$ **ass2**$(\langle x \rangle, \langle x+1 \rangle)$
x:=x+1

This law generates the following proof obligation

$$(x^2 \leq n \wedge 0 \leq x \wedge (x+1)^2 \leq n \wedge x = x_0) \Rightarrow$$
$$((x+1)^2 \leq n \wedge 0 \leq x+1 \wedge 0 \leq n - (x+1)^2 < n - {x_0}^2).$$

So, by applying the following tactic

tactic takeConjAsInv$(0 \leq x, (\langle x \rangle, \langle 0 \rangle), n - x^2)$;
(**law skipInt** $\boxed{;}$
($\boxed{\textbf{do}}$ **law assig2**$(\langle x \rangle, \langle x+1 \rangle)$$\boxed{\textbf{od}}$));

we get the following program

$x := 0;$
do $(x+1)^2 \leq n \rightarrow$
$x := x + 1$
od

8.1.3 Another Example

Sometimes we need to apply some laws before using the tactic. In this subsection we present an example of this kind.

$q, r : [a \geq 0 \wedge b > 0, q = a \operatorname{div} b \wedge r = a \operatorname{mod} b]$
$\sqsubseteq$ **strPost**$(a = q * b + r \wedge 0 \leq r \wedge r < b)$
$q, r : [a \geq 0 \wedge b > 0, a = q * b + r \wedge 0 \leq r \wedge r < b]$ $\lhd$

This law generates the following proof obligation

$$(a = q * b + r \wedge 0 \leq r \wedge r < b) \Rightarrow (q = a \operatorname{div} b \wedge r = a \operatorname{mod} b).$$

$\sqsubseteq$ **Tactic takeConjAsInv**$(b > 0, (\langle q, r \rangle, \langle 0, a \rangle), r < b)$ $\lhd$
$\sqsubseteq$ **strPost**$(a = q * b + r \wedge 0 \leq r \wedge r < b \wedge b > 0)$
$q, r : [a \geq 0 \wedge b > 0, a = q * b + r \wedge 0 \leq r \wedge r < b \wedge b > 0]$ $\lhd$

This law generates the following proof obligation

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

$\sqsubseteq$ **seqCom**$(a = q * b + r \wedge 0 \leq r \wedge b > 0)$
  $q, r : [a \geq 0 \wedge b > 0, a = q * b + r \wedge 0 \leq r \wedge b > 0];$ $\lhd$
  $q, r : [a = q * b + r \wedge 0 \leq r \wedge b > 0, a = q * b + r \wedge 0 \leq r \wedge b > 0 \wedge r < b]$ (i)
$\sqsubseteq$ **assig**$(\langle q, r\rangle, \langle 0, a\rangle)$
  q,r:=0,a
(i) $\sqsubseteq$ **iter**$(r \geq b, r)$
  do $r \geq b \rightarrow$
    $q, r : [a = q * b + r \wedge 0 \leq r \wedge b > 0 \wedge r \geq b,$
      $a = q * b + r \wedge 0 \leq r \wedge b > 0 \wedge 0 \leq r < r_0]$ $\lhd$
  od
$\sqsubseteq$ **assig2**$(\langle q, r\rangle, \langle q + 1, r - b\rangle)$
  q,r := q+1,r-b

This law generates the following proof obligation

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

So, by applying the following tactic

law strPost$(a = q * b + r \wedge 0 \leq r \wedge r < b)$;
tactic takeConjAsInv$(b > 0, (\langle q, r\rangle, \langle 0, a\rangle), r < b)$;
(**law skipInt** $\boxed{;}$
($\boxed{\textbf{do}}$ **law assig2**$(\langle q, r\rangle, \langle q + 1, r - b\rangle)$$\boxed{\textbf{od}}$))

we get the following program

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

8.2 Tactic replConsByVar

8.2.1 The Tactic

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**.

This tactic has five arguments: the variable declaration $newV : T$, the constant c, and the arguments taken by the tactic **takeConjAsInv**. It applies to any specification $w : [pre, post]$ to introduce an initialized iteration whose invariant is $post[newV/cons] \wedge invBound$ and whose variant is *variant*. The initialization is $lstVar := lstVal$. In this tactic the predicate *invBound* states the range limits of the new variable $newV$ which is used as a indexing variable of the iteration.

Tactic replConsByVar $(newV : T, cons, invBound, (lstVar, lstVal), variant)$
applies to $w : [pre, post]$
do **law varInt**$(newV : T)$;
$\boxed{\textbf{var}}$ **law strPost**$(post[newV/cons] \wedge newV = cons)$;
tactic takeConjAsInv$(invBound, (lstVar, lstVal), variant)$; $\boxed{]\!]}$
proof obligations
1.$(post[newV/cons] \wedge newV = cons) \Rightarrow post$
2.$(post[newV/cons] \wedge newV = cons \wedge invbound) \Rightarrow$
$(post[newV/cons] \wedge newV = cons)$
3.$pre \Rightarrow (post[newV/cons] \wedge invBound)[lstVal/lstVar]$
program generated
$[\![$**var** $newV : T \bullet$
$lstVar := lstVal$;
do $\neg\ newV = cons \rightarrow$
$newV, w : [post[newV/cons] \wedge invBound \wedge \neg\ newV = cons,$
$post[newV/cons] \wedge invBound \wedge variant]$
od
$]\!]$
end

This tactic first introduces the new variable(law **varInt**), then it strengthens the postcondition to substitute the constant by the new variable. Afterwards this tactic calls the tactic **takeConjAsInv** to introduce the iteration.The proof obligation 1 is generated by the law **strPost**, and the others are generated by the tactic **takeConjAsInv**.

8.2.2 Examples

$w : [n \geq 0, s = \sum_{i=0}^{n} exp(i)]$ $\lhd$
Tactic replConsByVar$(x : \mathbb{N}, n, 0 \leq x \leq n, (\langle s, x\rangle, \langle 0, 0\rangle), n - x)$
$\sqsubseteq$ **varInt**$(x : \mathbb{N})$
$[\![$**var** $x : \mathbb{N} \bullet$
$s, x : [n \geq 0, s = \sum_{i=0}^{n} exp(i)]$ $\lhd$
$]\!]$
$\sqsubseteq$ **strPost**$(s = \sum_{i=0}^{x} exp(i) \wedge x = n)$
$s, x : [n \geq 0, s = \sum_{i=0}^{x} exp(i) \wedge x = n]$ $\lhd$

This law generates the following proof obligation

$$(s = \sum_{i=0}^{x} exp(i) \wedge x = n) \Rightarrow (s = \sum_{i=0}^{n} exp(i))$$

$\sqsubseteq$ **Tactic takeConjAsInv**$(0 \leq x \leq n, (\langle s, x\rangle, \langle 0, 0\rangle), n - x)$
$\sqsubseteq$ **strPost**$(s = \sum_{i=0}^{x} exp(i) \wedge x = n \wedge 0 \leq x \leq n)$
$s, x : [n \geq 0, s = \sum_{i=0}^{x} exp(i) \wedge x = n \wedge 0 \leq x \leq n]$ $\lhd$

This law generates the following proof obligation

$$(s = \sum_{i=0}^{x} exp(i) \wedge x = n \wedge 0 \leq x \leq n) \Rightarrow \\ \quad (s = \sum_{i=0}^{x} exp(i) \wedge x = n)$$

$\sqsubseteq$ **seqCom**$(s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n)$
$\quad s, x : [n \geq 0, s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n]$ $\triangleleft$
$\quad s, x : [s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n, s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n \wedge x = n]$ (i)
$\sqsubseteq$ **assig**$(\langle s, x\rangle, \langle 0, 0\rangle)$
$\quad s, x := 0, 0$

This law generates the following proof obligation

$$n \geq 0 \Rightarrow (0 = \sum_{i=0}^{0} exp(i) \wedge 0 \leq 0 \leq n)$$

(i) $\sqsubseteq$ **iter**$(x \neq n, n - x)$
$\quad$ **do** $x \neq n \rightarrow$
$\quad\quad s, x : [s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n \wedge x \neq n,$
$\quad\quad\quad s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n \wedge 0 \leq n - x \leq n - x_0]$ $\triangleleft$
$\quad$ **od**
$\sqsubseteq$ **Tactic nextAndThisAssig**$(\langle x\rangle, \langle x+1\rangle, \langle s\rangle, \langle s + exp(x)\rangle)$
$\quad\sqsubseteq$ **fassig**$(\langle x\rangle, \langle x+1\rangle)$
$\quad s, x : [s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n \wedge x \neq n,$
$\quad\quad s = \sum_{i=0}^{x+1} exp(i) \wedge 0 \leq x + 1 \leq n \wedge 0 \leq n - (x+1) \leq n - x_0];$ $\triangleleft$
$\quad x := x + 1$
$\quad\sqsubseteq$ **assig2**$(\langle s\rangle, \langle s + exp(x)\rangle)$
$\quad s := s + exp(x)$

This law generates the following proof obligation

$$(x = x_0 \wedge s = \sum_{i=0}^{x} exp(i) \wedge 0 \leq x \leq n \wedge x \neq n) \Rightarrow \\ \quad (s + exp(x) = \sum_{i=0}^{x+1} exp(i) \wedge 0 \leq x + 1 \leq n \wedge 0 \leq n - (x+1) < n - x_0)$$

So, by applying the following tactic

tactic replConsByVar$(x : \mathbb{N}, n, 0 \leq x \leq n, (\langle s, x\rangle, \langle 0, 0\rangle), n - x)$;
$($$\boxed{\textbf{var}}$
$\quad$(**law skipInt** $\boxed{;}$
$\quad$($\boxed{\textbf{do}}$ **tactic nextAndThisAssig**$(\langle x\rangle, \langle x+1\rangle, \langle s\rangle, \langle s + exp(x)\rangle)$$\boxed{\textbf{od}}$))
$\boxed{]\!]}$)

We get the following program

$[\![$**var** $x : \mathbb{N} \bullet$
$\quad s, x := 0, 0;$
$\quad$ **do** $x \neq n \rightarrow$
$\quad\quad s := s + exp(x);$
$\quad\quad x := x + 1;$
$\quad$ **od**
$]\!]$

8.2.3 Another Example

$r : [a \geq 0 \wedge b \geq 0, r = a^b]$
$\sqsubseteq$ **replConsByVar**$(x : \mathbb{Z}, b, 0 \leq x \leq b, (\langle x, r\rangle, \langle 0, 1\rangle), b - x)$
  $\sqsubseteq$ **varInt**$(x : \mathbb{Z})$
    $[\![$**var** $x : \mathbb{Z} \bullet$
      $r, x : [a \geq 0 \wedge b \geq 0, r = a^b]$ $\lhd$
    $]\!]$
  $\sqsubseteq$ **strPost**$(r = a^x \wedge x = b)$
    $r, x : [a \geq 0 \wedge b \geq 0, r = a^x \wedge x = b]$ $\lhd$

This law generates the following proof obligation

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

  $\sqsubseteq$ **takeConjAsInv**$(0 \leq x \leq b, (\langle x, r\rangle, \langle 0, 1\rangle), b - x)$
    $\sqsubseteq$ **strPost**$(r = a^x \wedge x = b \wedge 0 \leq x \leq b)$
      $r, x : [a \geq 0 \wedge b \geq 0, r = a^x \wedge x = b \wedge 0 \leq x \leq b]$ $\lhd$

This law generates the following proof obligation

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

    $\sqsubseteq$ **seqCom**$(r = a^x \wedge 0 \leq x \leq b)$
      $r, x : [a \geq 0 \wedge b \geq 0, r = a^x \wedge 0 \leq x \leq b]$ $\lhd$
      $r, x : [r = a^x \wedge 0 \leq x \leq b, r = a^x \wedge 0 \leq x \leq b \wedge x = b]$ (i)
    $\sqsubseteq$ **assig**$(\langle x, r\rangle, \langle 0, 1\rangle)$
      x,r:=0,1;

This law generates the following proof obligation

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

  (i) $\sqsubseteq$ **iter**$(x \neq b, x - b)$
      do $x \neq b \rightarrow$
        $r, x : [r = a^x \wedge 0 \leq x \leq b \wedge x \neq b,$
          $r = a^x \wedge 0 \leq x \leq b \wedge 0 \leq b - x \leq b - x_0]$ $\lhd$
      od
$\sqsubseteq$ **assig2**$(\langle r, x\rangle, \langle r * a, x + 1\rangle)$
  $r, x := r * a, x + 1;$

This law generates the following proof obligation

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

So, by applying the following tactic

$$
\begin{array}{l}
\textbf{tactic replConsByVar}(x:\mathbb{Z}, b, 0 \leq x \leq b, (\langle x, r\rangle, \langle 0, 1\rangle), b - x);\\
(\boxed{\textbf{var}}\\
\quad (\textbf{law skipInt}\,\boxed{;}\\
\quad (\boxed{\textbf{do}}\,\textbf{law assig2}(\langle r, x\rangle, \langle r * a, x + 1\rangle)\boxed{\textbf{od}}))\\
\boxed{\rrbracket})
\end{array}
$$

We get the following program

$$
\begin{array}{l}
\llbracket \textbf{var } x : \mathbb{Z} \bullet\\
\quad x, r := 0, 1;\\
\quad \textbf{do } x \neq b \rightarrow\\
\quad\quad r, x := r * a, x + 1;\\
\quad \textbf{od}\\
\rrbracket
\end{array}
$$

8.3 Tactic alternation

8.3.1 The Tactic

Many times in a refinement we need to introduce an alternation and then to apply a tactic to each of its branches. With the purpose we define efforts the **tactic alternation**.

$$
\begin{array}{l}
\textbf{Tactic alternation}\\
\quad (\mathit{guardsList}, \mathit{tacticsList})\\
\textbf{applies to}\\
\quad w : [\mathit{pre}, \mathit{post}]\\
\textbf{do}\\
\quad \textbf{law alt}(\mathit{guardsList});\\
\quad \boxed{\textbf{if}}\,\mathit{tacticsList}\,\boxed{\textbf{fi}}\\
\textbf{proof obligations}\\
\quad \mathit{pre} \Rightarrow \bigvee(\mathit{guardsList})\\
\textbf{program generated}\\
\quad \textbf{if}(?i.\mathit{guardsList}_i \rightarrow (\mathit{tacticsList}_i\ P))\,\textbf{fi}
\end{array}
$$

This tactic takes as arguments two lists: the first one is the list of guards *guardsList* of the alternation and the second one the list of tactics *tacticsList* which are applied to each guarded command. It introduces an alternation in the program. Afterwards it applies each tactic of the *tacticsList* it takes as argument to each guarded command.

Some aplications of this tactic are shown in the following sections.

8.4 Tactic strengInv

8.4.1 The Tactic

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

The tactic **strengInv** has seven arguments: a variable declaration $newV1 : T1$, 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 whose invariant is $(post \wedge streng)[newV2/cons] \wedge invBound$ and variant is *variant*. The initialization is $lstVar := lstVal$. In this tactic the predicate *invBound* states the range limits (typically 0 and *cons*) of the new variable $newV2$ which is used as an indexing variable of the interation. The new variable $newV1$ 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
$\quad (newV1 : T1, streng, newV2 : T2, cons, invBound, (lstVar, lstVal), variant)$
applies to $w : [pre, post]$
do **law varInt**$(newV1 : T1)$;
$\quad \boxed{\textbf{var}}$ **law strPost**$(post \wedge streng)$;
$\qquad$ **tactic replConsByVar**$(newV2 : T2, cons, invBound, (lstVar, lstVal), variant)$;
$\quad \boxed{]\!]}$
proof obligations
$\quad$ 1.$(post \wedge streng) \Rightarrow post$
$\quad$ 2.$((pos \wedge streng)[newV2/cons] \wedge newV2 = cons) \Rightarrow (pos \wedge streng)$
$\quad$ 3.$((pos \wedge streng)[newV2/cons] \wedge newV2 = cons \wedge invBound) \Rightarrow$
$\qquad ((pos \wedge streng)[newV2/cons] \wedge newV2 = cons)$
$\quad$ 4.$pre \Rightarrow ((pos \wedge streng)[newV2/cons] \wedge invBound)[lstVal/lstVar]$
program generated
$\quad [\![$**var** $newV1 : T1 \bullet$
$\qquad [\![$**var** $newV2 : T2 \bullet$
$\qquad\quad lstVar := lstVal;$
$\qquad\quad$ **do** $\neg\ newV2 = cons \rightarrow$
$\qquad\qquad newV1, newV2, w : [(post \wedge streng)[newV2/cons] \wedge$
$\qquad\qquad\qquad invBound \wedge \neg\ newV2 = cons,$
$\qquad\qquad\qquad (post \wedge streng)[newV2/cons] \wedge$
$\qquad\qquad\qquad invBound \wedge variant]$
$\qquad\quad$ **od** $]\!]\ ]\!]$
end

The first step of this tactic is to introduce a new variable which3 is used to change (strenghten) the postcondition. Then, this tactic strengthens the postcondition and calls the tactic **replConsByVar**.

8.4.2 Examples

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

This law generates the following proof obligation

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

$$
\begin{array}{l}
\quad \sqsubseteq \textbf{replConsByVar}(m : \mathbb{Z}, n, 0 \leq m < n, (\langle m, r, s\rangle, \langle 0, 0, 0\rangle), n - m) \\
\qquad \sqsubseteq \textbf{varInt}(m : \mathbb{Z}) \\
\qquad\quad [\![\textbf{var}\, m : \mathbb{Z} \bullet \\
\qquad\qquad r, s, m : [n \geq 0, \\
\qquad\qquad\qquad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\}] \qquad \lhd \\
\qquad\quad]\!] \\
\qquad \sqsubseteq \textbf{strPost}((r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < n \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < n \wedge f[i] \leq 0\})[m/n] \wedge \\
\qquad\qquad m = n) \\
\qquad\quad r, s, m : [n \geq 0, \\
\qquad\qquad\quad r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad\quad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge m = n] \qquad \lhd
\end{array}
$$

This law generates the following proof obligation

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

$$
\begin{array}{l}
\sqsubseteq \textbf{takeConjAsInv}(0 \le m \le n, (\langle m, r, s\rangle, \langle 0,0,0\rangle), n-m) \\
\quad \sqsubseteq \textbf{strPost}(r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge \\
\qquad\qquad m = n \wedge 0 \le m \le n) \\
\qquad r,s,m : [n \ge 0, r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge m = n \wedge 0 \le m \le n] \qquad \lhd
\end{array}
$$

This law generates the following proof obligation

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

$$
\begin{array}{l}
\quad \sqsubseteq \textbf{seqCom}(r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge \\
\qquad\qquad 0 \le m \le n) \\
\qquad r,s,m : [n \ge 0, \\
\qquad\qquad\quad r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge 0 \le m \le n]; \qquad \lhd \\
\\
\qquad r,s,m : [r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge 0 \le m \le n, \\
\qquad\qquad\quad r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge 0 \le m \le n \wedge m = n]; \qquad (i) \\
\quad \sqsubseteq \textbf{assig}(\langle m, r, s\rangle, \langle 0,0,0\rangle) \\
\qquad m, s, r := 0, 0, 0
\end{array}
$$

This law generates the following proof obligation

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

$$
\begin{array}{l}
\text{(i)} \quad \sqsubseteq \textbf{iter}(m \ne n, n-m) \\
\qquad\quad \textbf{do}\ \ m \ne n \rightarrow \\
\qquad\qquad r,s,m : [r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge 0 \le m \le n \wedge m \ne n, \\
\qquad\qquad\qquad\quad r = \#\{i,j:\mathbb{N} \mid 0 \le i < j < m \wedge f[i] \le 0 \wedge f[j] \ge 0\} \wedge \\
\qquad\qquad\qquad\quad s = \#\{i:\mathbb{N} \mid 0 \le i < m \wedge f[i] \le 0\} \wedge 0 \le n-m < n-m_0]; \ \lhd \\
\qquad\quad \textbf{od}
\end{array}
$$

$$
\begin{array}{l}
\sqsubseteq \textbf{fassig}(\langle m\rangle, \langle m+1\rangle) \\
\quad r, s, m : [r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad 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 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 s = \#\{i : \mathbb{N} \mid 0 \lhd\leq i < m+1 \wedge f[i] \leq 0\} \wedge 0 \leq n-(m+1) < n - m_0]; \\
\quad m := m + 1 \\
\sqsubseteq \textbf{seqCom}(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 s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge 0 \leq m \leq n \wedge m \neq n) \\
\quad r, s, m : [r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad 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 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 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 \lhd \\
\\
\quad r, s, m : [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 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 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 s = \#\{i : \mathbb{N} \mid 0 \leq i < m+1 \wedge f[i] \leq 0\} \wedge 0 \leq n-(m+1) < n - m_0] \qquad (ii) \\
\sqsubseteq \textbf{alternation}((\langle f[m] < 0, f[m] \geq 0\rangle, \langle \textbf{skip}, \textbf{assig}(\langle r\rangle, \langle r+s\rangle)\rangle)) \\
\quad \sqsubseteq \textbf{alt}(\langle f[m] < 0, f[m] \geq 0\rangle) \\
\qquad \textbf{if } f[m] < 0 \rightarrow \\
\qquad\quad r, s, m : [f[m] < 0 \wedge r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad\qquad 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 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 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 \lhd \\
\qquad\quad [\,]\, f[m] \geq 0 \rightarrow \\
\qquad\quad r, s, m : [f[m] \geq 0 \wedge r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad\qquad 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 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 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 (iii) \\
\qquad \textbf{fi}
\end{array}
$$

This law generates the following proof obligation

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

$$
\begin{array}{l}
\sqsubseteq \textbf{skipInt} \\
\quad \textbf{skip}
\end{array}
$$

This law generates the following proof obligation

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

(iii) $\sqsubseteq$ **assig**$(\langle r\rangle, \langle r+s\rangle)$

$\quad r := r + s;$

This law generates the following proof obligation

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

(ii) $\sqsubseteq$ **alternation**$((\langle f[m] > 0, f[m] \leq 0\rangle, \langle \textbf{skip}, \textbf{assig2}(\langle s\rangle, \langle s+1\rangle)\rangle))$

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

$$
\begin{array}{l}
\textbf{if } f[m] > 0 \rightarrow \\
\quad r, s, m : [f[m] > 0 \wedge r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m+1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge 0 \leq m \leq n \wedge m \neq n, \\
\qquad\qquad r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m+1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m+1 \wedge f[i] \leq 0\} \wedge 0 \leq n - (m+1) < n - m_0]; \triangleleft \\
\;[]\, f[m] \geq 0 \rightarrow \\
\quad r, s, m : [f[m] \leq 0 \wedge r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m+1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge 0 \leq m \leq n \wedge m \neq n, \\
\qquad\qquad r = \#\{i,j : \mathbb{N} \mid 0 \leq i < j < m+1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge \\
\qquad\qquad s = \#\{i : \mathbb{N} \mid 0 \leq i < m+1 \wedge f[i] \leq 0\} \wedge 0 \leq n - (m+1) < n - m_0]; (iv) \\
\textbf{fi}
\end{array}
$$

This law generates the following proof obligation

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

$\sqsubseteq$ **skipInt**

$\quad$ **skip**

This law generates the following proof obligation

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

(iv) $\sqsubseteq$ **assig2**$(\langle s\rangle, \langle s+1\rangle)$

$\quad s := s + 1$

This law generates the following proof obligation

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

So, by applying the following tactic

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

$\boxed{\textbf{var}}$

$\boxed{\textbf{var}}$
skip $\boxed{;}$
$\boxed{\textbf{do}}$
law fassig$(\langle m\rangle, \langle m + 1\rangle);$
(**law seqCom**$(r = \#\{i, j : \mathbb{N} \mid 0 \leq i < j < m + 1 \wedge f[i] \leq 0 \wedge f[j] \geq 0\} \wedge$
$s = \#\{i : \mathbb{N} \mid 0 \leq i < m \wedge f[i] \leq 0\} \wedge 0 \leq m \leq n \wedge m \neq n);$
(**tactic alternation**$(\langle f[m] < 0, f[m] \geq 0\rangle, \langle$**skip**, **assig**$(\langle r\rangle, \langle r + s\rangle)\rangle)$ $\boxed{;}$
tactic alternation$(\langle f[m] > 0, f[m] \leq 0\rangle, \langle$**skip**, **assig2**$(\langle s\rangle, \langle s + 1\rangle)\rangle)))$ $\boxed{;}$
skip
$\boxed{\textbf{od}}$
$\boxed{]\!]}$
$\boxed{]\!]}$

we get the following program

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

8.5 Tactic tailInvariant

8.5.1 The Tactic

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 $lstVar_2 := lstVal_2$, 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 $lstVar_1 := lstVal_1$. Inside the iteration body this tactic includes an alternation.

This tactic has ten arguments: the first two arguments are variable declarations, which are used as indexing variables in the iteration. The next argument is a predicate $invBound$. It is used to make a conjunction with the next argument, $notGuard$, which is also a predicate and represents the negation of the guard of the iteration. The next argument is a pair $(lstVar_1, lstVal_1)$, where $lstVar_1$ is a list of variables and $lstVal_1$ is a list of values. They are used in the initialization of the iteration. An integer expression $variant$ is the next argument and represents the variant of the iteration. The next two arguments $guardsList$ and $tacticsList$ are used to call the tactic **alternation**, which generates an alternation with the guards given as argument and applies the tactics t_i of the list of tactics also given as argument to the corresponding guarded programs. Its definition is very simple and is omitted for the sake of conciseness. Finally we have a pair $(lstVar_2, lstVal_2)$, where $lstVar_2$ is a list of variables and $lstVal_2$ is a list of values. They are used in the final assignment after the iteration has finished.

Tactic tailInvariant

$(var1 : T1, var2 : T2, invConj, notGuard, invBound, (lstVar_1, lstVal_1),$
$variant, guardsList, tacticsList, (lstVar_2, lstVal_2))$

applies to $w : [pre, post]$

do **law varInt**$(var1 : T1, var2 : T2)$;

$\boxed{\textbf{var}}$ **law seqCom**$(invConj \wedge notGuard)$;

(**tactic takeConjAsInv**$(invBound, (lstVar_1, lstVal_1), variant)$;

(**skip**$\boxed{;}$ $\boxed{\textbf{do}}$ **tactic alternation**$(guardsList, tacticsList)$$\boxed{\textbf{od}}$))

$\boxed{;}$

law assig$(lstVar_2, lstVal_2)$; $\boxed{]\!]}$

proof obligations

1.$(invConj \wedge notGuard \wedge invBound) \Rightarrow (invConj \wedge notGuard)$

2. $pre \Rightarrow (invConj \wedge invBound)[lstVal/lstVar]$

3. $(invConj \wedge invBound \wedge \neg\, notGuard) \Rightarrow \bigvee guardsList$

4. $invConj \wedge notGuard \Rightarrow post[lstVal/lstVar]$

program generated

$[\![$**var** $var1 : T1$; **var** $var2 : T2 \bullet$

$lstVar_1 := lstVal_1$;

do $\neg\, notGuard \rightarrow$

if$(?i.guardsList_i \rightarrow$

$(tacticsList_i\ w : [guardsList_i \wedge invConj \wedge invBound \wedge \neg\, notGuard,$
$invConj \wedge invBound \wedge 0 \leq variant < variant_0]))$ **fi**

od

$lstVar_2 := lstVal_2$;

$]\!]$

end

The tactic **tailInvariant** first introduces two variables (arguments $var1 : T_1$ and $var2 : T_2$). 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 assignmen-t. The tactic applies the tactic **takeConjAsInv** to the first program using the $invBound$, $(lstVar_1, lstVal_1)$ and $variant$ arguments. The tactic also applies the **law assig** to the second program using the argument $(lstVar_2, lstVal_2)$. Inside the iteration body, the tactic applies the tactic **alternation** using the arguments $guardsList$ and $tacticsList$. The proof obligations 1 and 2 are generated by the tactic **takConjAsInv**. The proof obligation 3 is generated by the law **alt**, and finally the proof obligation 4 is generated by the law **assig**.

8.5.2 Examples

In this example we will use the following function

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

$r : [n \geq 0, r = max\, i : \mathbb{N} \mid 0 \leq i \leq n \bullet A.i]$ $\lhd$

$\sqsubseteq$ **strPost**$(r = F.0.n)$

$r = [n \geq 0, r = F.0.n]$ $\lhd$

This law generates the following proof obligation

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

$\sqsubseteq$ **tailInvariant**$(x : \mathbb{N}, y : \mathbb{N}, F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n, x = y, true,$
$(\langle x, y\rangle, \langle 0, n\rangle), y - x, \langle A.x \leq A.y, A.x > A.y\rangle,$
$\langle$**assig2**$(\langle x\rangle, \langle x + 1\rangle),$ **assig2**$(\langle y\rangle, \langle y - 1\rangle)\rangle)$

$\sqsubseteq$ **varInt**$(x : \mathbb{N}, y : \mathbb{N})$

$[\![$**var** $x : \mathbb{N};\ y : \mathbb{N} \bullet$

$y, x, r : [n \geq 0, r = F.0.n]$ $\lhd$

$]\!]$

$\sqsubseteq$ **seqCom**$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y)$

$y, x, r : [n \geq 0, F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y]$ $\lhd$

$y, x, r : [F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y, r = F.0.n]$ (i)

$\sqsubseteq$ **takeConjAsInv**$(true, \langle x, y\rangle, \langle 0, n\rangle), y - x)$

$\sqsubseteq$ **strPost**$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y \wedge true)$

$y, x, r : [n \geq 0, F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y]$ $\lhd$

This law generates the following proof obligation

$$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y \wedge true) \Rightarrow$$
$$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y)$$

$\sqsubseteq$ **seqCom**$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n)$

$y, x, r : [n \geq 0, F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n]$ $\lhd$

$y, x, r : [F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n,$
$F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x = y]$ (ii)

$\sqsubseteq$ **assig**$(\langle x, y\rangle, \langle 0, n\rangle)$

$x, y := 0, n;$

This law generates the following proof obligation

$$n \geq 0 \Rightarrow (F.0.n = F.0.n \wedge 0 \leq 0 \leq n \leq n)$$

(ii) $\sqsubseteq$ **iter**$(< \neg\, notGuard >, variant)$

do $x \neq y \rightarrow$

$y, x, r : [F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x \neq y,$
$F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge 0 \leq y - x < y_0 - x_0]$ $\lhd$

od

$\sqsubseteq$ **alternation**$(\langle A.x \leq A.y, A.x > A.y\rangle, \langle$**assig2**$(\langle x\rangle, \langle x+1\rangle),$ **assig2**$(\langle y\rangle, \langle y-1\rangle)\rangle)$

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

if $A.x \leq A.y \rightarrow$

$y, x, r : [F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x \neq y \wedge A.x \leq A.y,$
$F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge 0 \leq y - x < y_0 - x_0] \lhd$

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

$y, x, r : [F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x \neq y \wedge A.x > A.y,$
$F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge 0 \leq y - x < y_0 - x_0](iii)$

fi

This law generates the following proof obligation

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

$\sqsubseteq$ **assig2**$(\langle x\rangle, \langle x+1\rangle)$

x:=x+1;

This law generates the following proof obligation

$$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x \neq y \wedge A.x \leq A.y) \Rightarrow$$
$$(F.(x+1).y = F.0.n \wedge 0 \leq x+1 \leq y \leq n \wedge 0 \leq y - (x+1) \leq y_0 - x_0)$$

(iii) **assig2**$(\langle y\rangle, \langle y-1\rangle)$

y:=y-1;

This law generates the following proof obligation

$$(F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n \wedge x \neq y \wedge A.x \leq A.y) \Rightarrow$$
$$(F.x.(y-1) = F.0.n \wedge 0 \leq x \leq y-1 \leq n \wedge 0 \leq (y-1) + x \leq y_0 - x_0)$$

(i) $\sqsubseteq$ **assig**$(\langle r\rangle, \langle A.x\rangle)$

$r := A.x$

This law generates the following proof obligation

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

So, by applying the following tactic

strPost$(r = F.0.n)$;
tailInvariant$(x : \mathbb{N}, y : \mathbb{N}, F.x.y = F.0.n \wedge 0 \leq x \leq y \leq n, x = y, true,$
$(\langle x, y\rangle, \langle 0, n\rangle), y - x, \langle A.x \leq A.y, A.x > A.y\rangle,$
$\langle$**assig2**$(\langle x\rangle, \langle x+1\rangle),$ **assig2**$(\langle y\rangle, \langle y-1\rangle)\rangle)$

we get the following program

$$
\begin{array}{l}
[\![\mathbf{var}\, x : \mathbb{N};\ y : \mathbb{N} \bullet \\
\qquad x, y := 0, 0; \\
\qquad \mathbf{do}\ \ x \neq y \rightarrow \\
\qquad\qquad \mathbf{if} \\
\qquad\qquad\qquad\quad A.x \leq A.y \rightarrow x := x + 1; \\
\qquad\qquad\qquad [\!]\ \ A.x > A.y \rightarrow y := y - 1; \\
\qquad\qquad \mathbf{fi} \\
\qquad \mathbf{od} \\
\qquad r := A.x; \\
]\!]
\end{array}
$$

8.6 Tactic procNoArgs

8.6.1 The Tactic

The **tactic procNoArgs** 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** $procName = procBody \bullet p[procBody \setminus procName]]\!]$
> **end**

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

8.6.2 Examples

$$
\begin{array}{ll}
p, q, r : [p \leq q \leq r \wedge \lfloor p, q, r \rfloor = \lfloor p_0, q_0, r_0 \rfloor] & \lhd \\
\sqsubseteq \mathbf{law\ fassig}((< p, q >, < (p \sqcap q), (p \sqcup q) >)) & \\
\quad p, q, r : [(p \sqcap q) \leq (p \sqcup q) \leq r \wedge \lfloor (p \sqcap q), (p \sqcup q), r \rfloor = \lfloor p_0, q_0, r_0 \rfloor]; & \lhd \\
\quad p, q := (p \sqcap q), (p \sqcup q) & \\
\sqsubseteq \mathbf{law\ fassig}((< q, r >, < (q \sqcap r), (q \sqcup r) >)) & \\
\quad p, q, r : [(p \sqcap (q \sqcap r)) \leq (p \sqcup (q \sqcap r)) \leq (q \sqcup r) \wedge \lfloor (p \sqcap (q \sqcap r)), & \\
\qquad\quad (p \sqcup (q \sqcap r)), (q \sqcup r) \rfloor = \lfloor p_0, q_0, r_0 \rfloor]; & \lhd \\
\quad q, r := (q \sqcap r), (q \sqcup r) & \\
\sqsubseteq \mathbf{law\ assig2}((< p, q >, < (p \sqcap q), (p \sqcup q) >)) & \\
\quad p, q := (p \sqcap q), (p \sqcup q) &
\end{array}
$$

This law generates the following proof obligation

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

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

$\quad\sqsubseteq$ **law procNoArgsIntro**$(\mathbf{sort}, (\mathbf{p}, \mathbf{q} := \mathbf{p} \sqcap \mathbf{q}, \mathbf{p} \sqcup \mathbf{q}))$

$$
\begin{array}{l}
[\![\mathbf{proc}\ sort = p, q := p \sqcap q, p \sqcup q \bullet \\
\quad p, q := (p \sqcap q), (p \sqcup q); \\
\quad q, r := (q \sqcap r), (q \sqcup r); \\
\quad p, q := (p \sqcap q), (p \sqcup q)]\!]
\end{array}
$$

$\quad\sqsubseteq$ **law procNoParCall**

$$
\begin{array}{l}
[\![\mathbf{proc}\ sort = p, q := p \sqcap q, p \sqcup q \bullet \\
\quad sort;\ q, r := (q \sqcap r), (q \sqcup r);\ sort]\!]
\end{array}
$$

So, by applying the following tactic

$$
\begin{array}{l}
\mathbf{law\ fassig}((< p, q >, < (p \sqcap q), (p \sqcup q) >)); \\
(\mathbf{law\ fassig}((< q, r >, < (q \sqcap r), (q \sqcup r) >)) \boxed{;} \mathbf{skip}); \\
(\mathbf{law\ fassig}((< p, q >, < (p \sqcap q), (p \sqcup q) >)) \boxed{;} \mathbf{skip} \boxed{;} \mathbf{skip}); \\
\mathbf{tactic\ procNoArgs}(sort, (p, q := p \sqcap q, p \sqcup q))
\end{array}
$$

we get the following program

$$
\begin{array}{l}
[\![\mathbf{proc}\ sort = p, q := p \sqcap q, p \sqcup q \bullet \\
\quad sort;\ q, r := (q \sqcap r), (q \sqcup r);\ sort]\!]
\end{array}
$$

8.7 Tactic procCalls

8.7.1 The Tactic

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 **k** $v : T$ where **k** defines how the argument is 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, this tactic 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 callByValue1**$(head'\ \ args, head'\ \ pars)$;
  $\boxed{\textbf{val}}$ **tactic procCalls**$(tail\ \ pars, tail\ \ args)$) |
 (**law callByValue2**$(head'\ \ args, head'\ \ pars)$;
  ($\boxed{\textbf{val}}$ **law hideFrame**$(head'\ \ pars)$);
   tactic procCalls$(tail\ \ pars, tail\ \ args)$) |
 (**law callByResult**$(head'\ \ args, head'\ \ pars)$;
  $\boxed{\textbf{res}}$ **tactic procCalls**$(tail\ \ pars, tail\ \ args)$ |
 (**law callByValueResult**$(head'\ \ args, head'\ \ pars)$;
  $\boxed{\textbf{val\_res}}$ **tactic procCalls**$(tail\ \ pars, tail\ \ args)$) |
 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. 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 below.

8.8 Tactic procArgs

8.8.1 The Tactic

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 argument *pars* is a (possibly multiple) parameter declaration. We make use of the function *seqToList* to convert *args*, a ;-separated sequence of arguments declarations, to a list of declarations.

Tactic procArgs $(procName, pars, body, args)$
applies to p
do **law procArgsIntro**$(procName, pars, body)$;
 $\boxed{\textbf{pmain}}$
  tactic procCalls$(seqToList\ \ pars, args)$; $exhaust$(**law multiArgs**) $\boxed{]\!]}$;
 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.

8.9 Examples

$x : [0 \leq x, x^2 = x_0]$
$\sqsubseteq$ **tactic procArgs**$(sqrts, (\textbf{val}\ a : \mathbb{R};\ \textbf{res}\ b : \mathbb{R}), b : [0 \leq a, b^2 = a], < x, x >)$
  $\sqsubseteq$ **law procArgsIntro**$(sqrts, (\textbf{val}\ a : \mathbb{R};\ \textbf{res}\ b : \mathbb{R}), b : [0 \leq a, b^2 = a])$
    $[\![\textbf{proc}\ sqrts = (\textbf{val}\ a : \mathbb{R};\ \textbf{res}\ b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a]) \bullet$
      $x : [0 \leq x, x^2 = x_0]$ $\lhd$
    $]\!]$
  $\sqsubseteq$ **tactic procCalls**$(< \textbf{val}\ a : \mathbb{R}, \textbf{res}\ b : \mathbb{R} >, < x, x >)$
    $\sqsubseteq$ **law callByValue2**(x, a)
      $(\textbf{val}\ a : \mathbb{R} \bullet x, a : [0 \leq a, x^2 = a_0])(x)$ $\lhd$
    $\sqsubseteq$ **law hideFrame**(a)
      $(\textbf{val}\ a : \mathbb{R} \bullet x : [0 \leq a, x^2 = a])(x)$ $\lhd$
    $\sqsubseteq$ **tactic procCalls**$(< \textbf{res}\ b : \mathbb{R} >, < x >)$
      $\sqsubseteq$ **law callByResult**(x, b)
        $(\textbf{res}\ b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a])(x)$ $\lhd$
      $\sqsubseteq$ **tacticprocCalls**$(<>, <>)$
        $(\textbf{res}\ b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a])(x)$
  $\sqsubseteq$ *exhaust*(**law multiArgs**)
    law multiArgs
      $(\textbf{val}\ a : \mathbb{R}, \textbf{res}\ b : \mathbb{R} \bullet b : [0 \leq a, b^2 = a])(x, x)$ $\lhd$
  $\sqsubseteq$ **law procArgsCall**
    sqrts(x,x)

So, by applying the following tactic

tactic procArgs
$(sqrts, (\textbf{val}\ a : \mathbb{R};\ \textbf{res}\ b : \mathbb{R}), b : [0 \leq a, b^2 = a], < x, x >)$

we get the following program

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

8.10 Tactic recProcArgs

8.10.1 The Tactic

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
$(procName, args, variantName, variantExp, body, varsProcF, guardsList, tacticsList)$
applies to p
do **law variantIntro**$(procName, args, variantName, variantExp, body)$;
$\boxed{\textbf{pmainvariant}}$
tactic procCalls$(seqToList\ args, varsProcF)$;
$exhaust$(**law multiArgs**); $\boxed{]\!]}$;
law procArgsVariantBlockCall;
$\boxed{\textbf{pbodyvariant}}$
law absAssump; **tactic alternation**$(guardsList, tacticsList)$; $\boxed{]\!]}$;
law recursiveCall
end

This tactic first introduces a variant block using the law **variantIntro**. Then, as in the previous tactic, it uses the law **procCalls** and 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. The next step of this tactic is to refine the body of the procedure. First, the tactic uses the law **absAssump** to move the assumption that defines the variant, to the precondition of the specification. Then, the tactic applies the tactic **alternation**. Finally the tactic uses the law **recursiveCall** to introduce a recursive call to the program.

8.10.2 Examples

$f : [f = n! * 1]$ $\lhd$
$\sqsubseteq$ **Tactic recProcArgs**
$(fact,$ **val** $m, k : \mathbb{N}, V, m, (f : [f = m! * k]), < n, 1 >, < m = 0, m > 0 >,$
$<$ **law assig**$(f := k)$,
law strPos$(f = (m-1)! * m * k)$;
law callByValue1$(< m, k >, < m-1, m * k >)$;
$\boxed{\textbf{val}}$ (**law weakPre**$(0 \leq m < V)$;
law absAssump) $>$)
$\sqsubseteq$ **law variantIntro**$(fact,$ **val** $m, k : \mathbb{N}, V, m, (f : [f = m! * k]))$
$[\![$**proc** $fact = ($**val** $m, k : \mathbb{N} \bullet \{V = m\} f : [f = m! * k])$
variant V **is** $m \bullet$
$f : [f = n! * 1]$ $\lhd$
$]\!]$

$\sqsubseteq$ **tactic procCalls**$(< m, k >, < n, 1 >)$

$\sqsubseteq$ **law callByValue1**$(m, k : \mathbb{N}, < n, 1 >)$

$[\![$**proc** $fact = ($**val** $m, k : \mathbb{N} \bullet \{V = m\} f : [f = m! * k])$

variant V **is** $m \bullet$

$($**val** $m, k : \mathbb{N} \bullet f : [f = m! * k])(n, 1)$

$]\!]$ $\triangleleft$

$\sqsubseteq$ **law procArgsVariantBlockCall**

$[\![$**proc** $fact = ($**val** $m, k : \mathbb{N} \bullet \{V = m\} f : [f = m! * k])$ $\triangleleft$

variant V **is** $m \bullet$

$fact(n, 1)$

$]\!]$

This law generates the following proof obligation

$$\{V = m\} f : [f = m! * k] \sqsubseteq \{V = m\} f : [f = m! * k]$$

$\sqsubseteq$ **law absAssump**

$f : [V = m, f = m! * k]$ $\triangleleft$

$\sqsubseteq$ **tactic alternation**

$(< m = 0, m > 0 >,$

$<$ **law assig**$(f := k)$,

law strPos$(f = (m - 1)! * m * k)$;

law callByValue1$(< m, k >, < m - 1, m * k >)$;

$\boxed{\textbf{val}}$ (**law weakPre**$(0 \leq m < V)$;

law absAssump$) >)$

$\sqsubseteq$ **law alt**$(< m = 0, m > 0 >)$

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

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

$\sqsubseteq$ **law assig**$(f := k)$

$f := k$

The following proof obligation is generated in this case

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

(i) $\sqsubseteq$ **law strPos**$(f = (m - 1)! * m * k)$

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

We have the following proof obligation

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

$\sqsubseteq$ **law callByValue1**$(< m, k >, < m-1, m*k >)$
$(\mathbf{val}\ m, k : \mathbb{N} \bullet f : [V = m+1 \wedge m+1 > 0, f = m! * k])(m-1, m*k)$ $\lhd$
$\sqsubseteq$ **law weakPre**$(0 \leq m < V)$
$f : [0 \leq m < V, f = m! * k])$ $\lhd$

This law generates the following proof obligation

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

$\sqsubseteq$ **law assumpAbs**
$\{0 \leq m < V\} f : [f = m! * k]$

Now we have the following program

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

Which is refined as seen below

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

This law aplication generates the following proof obligation

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

So, by applying the following tactic

$$
\begin{array}{l}
\textbf{Tactic recProcArgs} \\
\quad (fact, \textbf{val}\ m, k : \mathbb{N}, V, m, f : [f = m! * k], < n, 1 >, < m = 0, m > 0 >, \\
\qquad < \textbf{law assig}(f := k), \\
\qquad\quad \textbf{law strPos}(f = (m-1)! * m * k); \\
\qquad\quad \textbf{law callByValue1}(< m, k >, < m-1, m * k >); \\
\qquad\quad \boxed{\textbf{val}}\ (\textbf{law weakPre}(0 \leq m < V); \\
\qquad\quad \textbf{law assumpAbs}) >)
\end{array}
$$

we get the following program

$$
\begin{array}{l}
[\![\\
\quad \textbf{proc}\ fact = (\textbf{val}\ m, k : \mathbb{N} \bullet \\
\qquad\quad \textbf{if}\ m = 0 \rightarrow f := k) \\
\qquad\qquad [\!]\ m > 0 \rightarrow fact(m-1, m * k) \\
\qquad\quad \textbf{fi} \\
\quad \textbf{variant}\ V\ \textbf{is}\ m \bullet \\
\quad fact(n, 1) \\
]\!]
\end{array}
$$

9 Acknowledgements

We thank Ana Cavalcanti from the LMF/CIn (Laboratory of Formal Methods of the Centro de Informática of the Universidade Federal de Pernambuco) for making format and content suggestions to improve this monograph.

10 Conclusion

We have presented RTL, a language suitable for the definition of refinement tactics. Using RTL we can specify commonly used strategies of program development and use them as transformation rules. This not only shortens the developments, but also improves their readability.

Then, we have defined the semantics of RTL. This work was based on Angel semantics. The difference is that Angel is a very general language and its goals have no defined structure. For us, a goal is called *RMCell*, 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 *RMCell* returns a list of *RMCells* which is the possible output programs with their corresponding set of proof obligations.

We have shown the soundness of algebraic laws for reasoning about RTL tactics. We covered most of the laws that have been proposed for Angel. For the vast majority of them, a proof is not available in the literature and is provided here in the context of RTL. In this monograph, the proofs of the laws and lemmas consider only finite lists. The extension of this result, considering infinite lists, is left as future work.

Since these laws are valid, the strategy proposed to reduce finite Angel tactics to a normal form can be applied to RTL tactics. We have expanded the proofs of the Lemmas and Theorems involved, exemplifying some of the procedures.

Generalising the normal form so to emcompass the rest of the language is not trivial and is still to be done. Basically, there are three points to conclude this task. We must include the tactic **tactic** t to the language. The idea is to substitute the tactic body where this tactic is applied. The other point consists of including recursion and the structural combinators. We have to extend the normal form to comtenplate these. In the case of structural combinators, we believe we should allow their ocurrence wherever a rule is allowed in the presented normal form, and require that the tactic in the structural combinators are already in the normal form.

We have defined a number of refinement tactics that capture standard strategies for the development of iterations and more recent formalizations of strategies for the development of procedural abstractions. These strategies have already been put forward in the literature mainly by means of examples, but as far as we know, they have never been formalized and expressed as a transformation rule.

Our language is independent of any particular tool or programming language. On the other hand, the LMF/CIn (Laboratory of Formal Methods of the Centro de Informática of the Universidade Federal de Pernambuco) is developing a tool to support the refinement calculus [3] and we intend to extend it to provide support for RTL. The refinement tactics presented here are going to compose a basic repository of tactics. The user is going to be able to apply these tactics as well as define new ones.

As already mentioned, to our knowledge, the only other work on refinement tactics is that presented in [7], where basically Prolog is used as a tactic language. It is not possible to define a tactic recursively and there are no structural combinators. Also, only a few examples of simple tactics are provided there.

A Definitions

The functions on lists and pairs used to define some constructors of the RTL language are defined in this section. These definitions are important, even if though they are well-known, because we make a very heavy use of their properties.

The list concatenation definition is written below:

$$
\begin{aligned}
&\langle\rangle \frown xs = xs \\
&(x : xs) \frown ys = x : (xs \frown ys)
\end{aligned}
$$

The distributed concatenation definition is written below:

$$
\begin{aligned}
&\frown / \langle\rangle = \langle\rangle \\
&\frown /(x : xs) = x \frown (\frown / xs)
\end{aligned}
$$

The function *length* returns the length of a given list and its definition is written below:

$$
\begin{aligned}
&length \ \langle\rangle = 0 \\
&length \ (x : xs) = 1 + (length \ xs)
\end{aligned}
$$

The $\prod$ is the distributed cartesian product is defined using a list cross product:

$$\prod < xs >= e2l * xs$$
$$\prod(xs : xss) =< (a : as) \mid a < -xs, as < -\prod xss >$$

$$e2l\ x =< x >$$

The function $\circ$ is also defined below:

$$<>^\circ\ n =<>$$
$$(f : fs)^\circ n = (f\ \ n) : (fs^\circ n)$$

The function apply is defined as

$$apply\ \ <>\ <>=<>$$
$$apply\ (f : fs)\ (x : xs) = (f\ \ x) : (apply\ fs\ xs)$$

The *head* and the *tail* function are also defined below. They are a version of the standard *head* and *tail* functions:

$$head\ <>=<>$$
$$head(x : xs) = x$$

$$tail\ <>=<>$$
$$tail(x : xs) = xs$$

The function $head'$ has as its definition

$$head'\ <>=<>$$
$$head'(x : xs) =< x >$$

Pairs in the form (a, b) are also used in the semantics of RTL. And the functions *fst* and *snd* are used in this monograph.

$$fst(a, b) = a$$
$$snd(a, b) = b$$

For a total function $f : A \rightarrow B$, $f* : A* \rightarrow B*$ is the function that operates on lists by applying f to each of their elements. Its definition is

$$f*\ <>=<>$$
$$f * (x : xs) = f(x) : (f *\ \ xs)$$

The operator $\bullet$ is used to compose functions and its definition is

$$(f \bullet g)\ x = f(g(x))$$

The operator $\oplus$ acts over functions and is the sobreposition. Its definition is

$$A \oplus B = x \mid x \in A \wedge x \notin B \cup x \mid x \notin B$$

The function *pair* takes two lists and returns a list of pairs with the first element from the first list and the second element from the corresponding element of the second list. Its definition is

$$pair\ x\ \ y = (x, y)$$

The function *add*, given a set of proof obligations and a RMCell, returns an *RMCell* which has as its proofs obligations the union of the set of proof obligations given as argument and the set of the proof obligations of the *RMCell* given as argument. It is defined below.

$$add\ \ pr_2\ (p, pr_1) = (p, pr_1 \cup pr_2)$$

B Laws of Refinement Calculus

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 assig$(\langle w \rangle, \langle E \rangle)$ $w : [pre, post] \sqsubseteq w := E$ provided $pre \Rightarrow post[w \setminus E]$

Law assig2$(\langle w \rangle, \langle E \rangle)$ $w, x : [pre, post] \sqsubseteq w := E$ provided $(w = w_0) \wedge pre \Rightarrow post[w \setminus E]$

Law fassig$(\langle x \rangle, \langle E \rangle)$For any term E $w, x : [pre, post] \sqsubseteq w, x : [pre, post[x \setminus E]];\ x := E$

Law seqCom(mid) For any mid $w : [pre, post] \sqsubseteq w : [pre, mid];\ w : [mid, post]$

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 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** $(?\ i.G_i \rightarrow w : [inv \wedge G_i, 0 \leq V \leq V[w \setminus w_0]])$ **od**

Law varInt$(n : T)$ $w : [pre, post] \sqsubseteq [\![$**var** $x : T \bullet w, x : [pre, post]]\!]$ provided x doen not occurs in w, pre, and $post$

Law skipIntro $w, x : [pre, post] \sqsubseteq$ **skip** provided $w = w_0 \wedge pre \Rightarrow post$

Law procNoArgsIntro(pn, p_1) $p_2 = [\![$**proc** $pn = p_1 \bullet p_2]\!]$ provided pn is not free in p_2

Law procNoArgsCall $[\![$**proc** $pn = p_1 \bullet p_2[p_1]]\!] = [\![$**proc** $pn = p_1 \bullet p_2[pn]]\!]$

Law procArgsIntro(pn, par, p_1) $p_2 = [\![$**proc** $pn = (par \bullet p_1) \bullet p_2]\!]$ provided pn is not free in p_2

Law procArgsCall
$[\![$**proc** $pn = (par \bullet p_1) \bullet p_2[(par \bullet p_1)(a)]]\!] = [\![$**proc** $pn = (par \bullet p_1) \bullet p_2[pn(a)]]\!]$

Law callByValue1(f, a) $w : [pre[f \setminus a], post[f, f_0 \setminus a, a_0]] = ($**val** $f \bullet w : [pre, post])(a)$ provided f is not in w and w is not free in a

Law callByValue2(f, a) $w : [pre[f \setminus a], post[f_0 \setminus a_0]] = ($**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 and f_0 is not free in $post$

Law callByValueResult(f, a)
$w, a : [pre[f \setminus a], post[f_0 \setminus a_0]] = (\mathbf{val\_res}\ f \bullet w, f : [pre, post[a \setminus f]])(a)$
provided f is not in w, and is not free $post$

Law hideFrame(x) $w, x : [pre, post] \sqsubseteq w : [pre, post[x_0 \setminus x]]$

Law variantIntro$(pr, pars, n, e, p_1)$
$p_2 =]\!]$ $\mathbf{proc}\, pr = (par \bullet \{n = e\}p_1)\,\mathbf{variant}\ \ n\ \ \mathbf{is}\ \ e \bullet p_2[\![$ provided pr and n are not free in e and p_2

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 procVariantBlockCall $[\![\mathbf{proc}\ \ pr = (par \bullet p_1)\,\mathbf{variant}\ n\,\mathbf{is}\ e \bullet p_2[(par \bullet p_3)(a)]]\!]$
$= [\![\mathbf{proc}\ \ pr = (par \bullet p_1)\,\mathbf{variant}\ n\,\mathbf{is}\ e \bullet p_2[(pr)(a)]]\!]$
provided pr is not recursive, n is not free in e and p_3, and $\{n = e\}p_3 \sqsubseteq p_1$

Law absAssump $\{pre'\}w : [pre, post] = w : [pre' \wedge pre, post]$

Law recursiveCall $[\![\mathbf{proc}\ \ pr = (par \bullet p_1[(par \bullet \{0 \leq e < n\}p_3)(a)])\,\mathbf{variant}\ n\,\mathbf{is}\ e \bullet p_2]\!]$
$= [\![\mathbf{proc}\ \ pr = (par \bullet p_1[pr(a)])\,\mathbf{variant}\ n\,\mathbf{is}\ e \bullet p_2]\!]$ provided n is not free in p_3 and $p_1[pr(a)]$, and $\{n = e\}p_3 \sqsubseteq p_1[(par \bullet \{0 \leq e < n\}p_3)(a)]$

C Proofs of the Laws

In this Appendix we prove the laws presented in this monograph. The lemmas used here are presented and proved in the Appendix D.

C.1 Basic properties of composition

Law 1.(a) **skip** ; $t = t$

Proof

$[\![\mathbf{skip};\ t]\!]\Gamma_L\,\Gamma_T\ r$	
$= \frown/ \bullet ([\![t]\!]\Gamma_L\,\Gamma_T)* \bullet [\![\mathbf{skip}]\!]\Gamma_L\,\Gamma_T\ r$	[Definition of ;]
$= \frown/(([\![t]\!]\Gamma_L\,\Gamma_T) * ([\![\mathbf{skip}]\!]\Gamma_L\,\Gamma_T\ r))$	[Functional Composition]
$= \frown/(([\![t]\!]\Gamma_L\,\Gamma_T) * \langle r \rangle)$	[Definition of **skip**]
$= \frown/ < [\![t]\!]\Gamma_L\,\Gamma_T\ r >$	[Definition of *]
$= [\![t]\!]\Gamma_L\,\Gamma_T\ r$	[Property of $\frown/$]

□

Law 1.(b) $t = \textbf{skip};\ t$

Proof

$[\![\textbf{skip};\ t]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![t]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\textbf{skip}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= \frown/([\![t]\!]\Gamma_L\ \Gamma_T) * (([\![\textbf{skip}]\!]\Gamma_L\ \Gamma_T\ r))$ [Functional Composition]
$= \frown/([\![t]\!]\Gamma_L\ \Gamma_T) * \langle r\rangle$ [Definition of **skip**]
$= \frown/ < [\![t]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of *]
$= [\![t]\!]\Gamma_L\ \Gamma_T\ r$ [Property of $\frown/$]

$\square$

Law 2.(a) $t \mid \textbf{fail} = t$

Proof

$[\![t \mid \textbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$
$= (\frown/ \bullet < ([\![t]\!]\Gamma_L\ \Gamma_T), ([\![\textbf{fail}]\!]\Gamma_L\ \Gamma_T) >^\circ)\ r$ [Definition of |]
$= \frown/ < [\![t]\!]\Gamma_L\ \Gamma_T\ r, [\![\textbf{fail}]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of °]
[Functional Composition]
$= \frown/ < [\![t]\!]\Gamma_L\ \Gamma_T\ r, <> >$ [Definition of **fail**]
$= [\![t]\!]\Gamma_L\ \Gamma_T\ r$ [Property of $\frown/$]

$\square$

Law 2.(b) $t = \textbf{fail} \mid t$

Proof

$([\![t]\!]\Gamma_L\ \Gamma_T)\ r$
$= \frown/ << >, ([\![t]\!]\Gamma_L\ \Gamma_T) >$ [Property of $\frown/$]
$= \frown/ < ([\![\textbf{fail}]\!]\Gamma_L\ \Gamma_T)\ r, ([\![t]\!]\Gamma_L\ \Gamma_T)\ r >$ [Definition of **fail**]
$= \frown/ \bullet < ([\![\textbf{fail}]\!]\Gamma_L\ \Gamma_T), ([\![t]\!]\Gamma_L\ \Gamma_T) >^\circ\ \ r$ [Definition of °]
[Functional Composition]
$= [\![\textbf{fail} \mid t]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of |]

$\square$

Law 3.(a) $t;\ \mathbf{fail} = \mathbf{fail}$

Proof

$[\![t;\ \mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of ;]
$= \frown/([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T) * ([\![t]\!]\Gamma_L\ \Gamma_T\ r)$ [Functional Composition]
$= \frown/ <<>>^{length([\![t]\!]\Gamma_L\ \Gamma_T)}$ [Definition of **fail** and $*$]

where

$$< lst >^0 = <>$$
$$< lst >^n = lst : (< lst >^{n-1})$$

$= <>$ [Definition of $\frown/$]
$= ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of **fail**]

$\square$

Law 3.(b) $\mathbf{fail} = t;\ \mathbf{fail}$

$[\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r = <>$ [Definition of **fail**]
$= \frown/ <<>>^{length([\![t]\!]\Gamma_L\ \Gamma_T)}$ [Definition of $< lst >^n$]
[Property of $\frown/$]
$= \frown/([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T) * ([\![t]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of **fail** and $*$]
$= \frown/ \bullet ([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t]\!]\Gamma_L\ \Gamma_T)\ r$ [Functional Composition]
$= [\![t;\ \mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]

$\square$

Law 4. $t_1 \mid (t_2 \mid t_3) = (t_1 \mid t_2) \mid t_3$

Proof

$[\![t_1 \mid (t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![(t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T >^\circ\ r$ [Definition of $\mid$]
[Functional Composition]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![(t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\frown/$]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown (\frown/(< [\![t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T >^\circ\ r))$ [Definition of $\mid$]
[Functional Composition]
$= ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown (([\![t_2]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\frown/$]
$= (([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)) \frown ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Property of $\frown$]

$= (\frown/ < ([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![t_2]\!]\Gamma_L\ \Gamma_T) >^\circ\ \ r) \frown ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\frown/$ and $\circ$]
$= ([\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\mid$]
[Functional Composition]
$= \frown/ < [\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T >^\circ\ \ r$ [Definition of $\frown/$ and $\circ$]
$= [\![(t_1 \mid t_2) \mid t_3]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of $\mid$]

$\square$

Law 5. $t_1;(t_2;t_3) = (t_1;t_2);t_3$

Proof

$([\![t_1;\ (t_2;\ t_3)]\!]\Gamma_L\ \Gamma_T))$
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![(t_1;\ t_2)]\!]\Gamma_L\ \Gamma_T)$ [Definition of ;]
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet (\frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T))$ [Definition of ;]
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet \frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$ [Property of $\bullet$]
$= \frown/ \bullet \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T) ** \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$ [Lemma D.1]
$= \frown/ \bullet \frown/* \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T) ** \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$ [Lemma D.2]
$= \frown/ \bullet (\frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T))* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$ [Property of $\bullet$ and $*$]
$= \frown/ \bullet (([\![t_2]\!]\Gamma_L\ \Gamma_T);\ ([\![t_3]\!]\Gamma_L\ \Gamma_T))* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T)$ [Definition of ;]
$= ([\![t_1;\ (t_2;\ t_3)]\!]\Gamma_L\ \Gamma_T)$ [Definition of ;]

$\square$

Law 6. $(t_1 \mid t_2);\ t_3 = (t_1;\ t_3) \mid (t_2;\ t_3)$

Proof

$[\![(t_1 \mid t_2);\ t_3]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet [\![(t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet \frown/ \bullet < [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T >^\circ\ \ r$ [Definition of |]
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet \frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of $\circ$]
$= \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $\frown/$]
$= \frown/ \bullet (([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) \frown (([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$
[Property of $*$]
$= (\frown/ \bullet (([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) \frown (\frown/ \bullet (([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_2]\!]\Gamma_L\ \Gamma_T)\ r)$
[Property of $\frown/$]
$= ([\![(t_1;\ t_3)]\!]\Gamma_L\ \Gamma_T)\ r \frown ([\![(t_2;\ t_3)]\!]\Gamma_L\ \Gamma_T)\ r$ [Definition of ;]
$= \frown/ \bullet < [\![(t_1;\ t_3)]\!]\Gamma_L\ \Gamma_T, [\![(t_2;\ t_3)]\!]\Gamma_L\ \Gamma_T >^\circ\ \ r$ [Definition of $\frown/$ and $\circ$]
$= [\![(t_1;\ t_3) \mid (t_2;\ t_3)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of |]

$\square$

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

Proof

$[\![t_1;\ (t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![t_2 \mid t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of ;]
$= \frown/ \bullet (\frown/ \bullet < [\![t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T >^\circ)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]
$= \frown/ \bullet < \frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, \frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) >$
[Definition of $^\circ$ and $*$]
[Property of $*$]
$= \frown/ \bullet < [\![t_1;\ t_2]\!]\Gamma_L\ \Gamma_T\ r, [\![t_1;\ t_3]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of ;]
$= \frown/ \bullet < [\![t_1;\ t_2]\!]\Gamma_L\ \Gamma_T, [\![t_1;\ t_3]\!]\Gamma_L\ \Gamma_T >^\circ\ r$ [Definition of $^\circ$]
$= [\![(t_1;\ t_2) \mid (t_1;\ t_3)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]

□

C.2 Laws involving Cut

Law 8. $!\,\mathbf{skip} = \mathbf{skip}$

Proof

$[\![!\,\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r$
$= head' \bullet [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]
$= head' < r >$ [Definition of **skip**]
$= < r >$ [Definition of *head′*]
$= [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of **skip**]

□

Law 9. $!\,\mathbf{fail} = \mathbf{fail}$

Proof

$[\![!\,\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$
$= head' \bullet [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]
$= head' <>$ [Definition of **fail**]
$= <>$ [Definition of *head′*]
$= [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of **fail**]

□

Law 10. $!(\textbf{law}\ \ l\ (args)) = \textbf{law}\ \ l\ (args)$

$[\![!\,\textbf{law}\ \ l\ (args)]\!]\Gamma_L\ \Gamma_T\ r$
$= head' \bullet ([\![\textbf{law}\ \ l\ (args)]\!]\Gamma_L\ \Gamma_T\ r)$ [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.3]
[Definition of $head'$]

$\square$

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

Proof.

$[\![!t_1;\ (t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T\ r$
$= ([\![(t_2 \mid t_3)]\!]\Gamma_L\ \Gamma_T) * \bullet\ [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= (\frown/ \bullet < [\![t_2]\!]\Gamma_L\ \Gamma_T, [\![t_3]\!]\Gamma_L\ \Gamma_T >^\circ) * \bullet\ [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of $|$]
$= (\frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T) * ([\![!t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_3]\!]\Gamma_L\ \Gamma_T) * ([\![!t_1]\!]\Gamma_L\ \Gamma_T\ r)$
[Definition of $^\circ$ and $*$]
$= (\frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T) * ([\![!t_1]\!]\Gamma_L\ \Gamma_T\ r)) \frown (\frown/ \bullet ([\![t_3]\!]\Gamma_L\ \Gamma_T) * ([\![!t_1]\!]\Gamma_L\ \Gamma_T\ r))$
[Lemma D.4]
$= ([\![!t_1;\ t_2]\!]\Gamma_L\ \Gamma_T) \frown ([\![!t_1;\ t_3]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of ;]
$= \frown/ \bullet < [\![!t_1;\ t_2]\!]\Gamma_L\ \Gamma_T, [\![!t_1;\ t_3]\!]\Gamma_L\ \Gamma_T >^\circ\ \ r$ [Definition of $\frown/$ and $^\circ$]
$= [\![(!t_1;\ t_2) \mid (!t_1;\ t_3)]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of $|$]

$\square$

Law 12. $!t_1;\ !t_2 =!(!t_1;\ !t_2)$

Proof

$[\![!t_1;\ !t_2]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![!t_2]\!]\Gamma_L\ \Gamma_T) * \bullet\ [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]
$= \frown/ \bullet ([\![!t_2]\!]\Gamma_L\ \Gamma_T) * \bullet\ head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= \frown/ \bullet ([\![!t_2]\!]\Gamma_L\ \Gamma_T) * \bullet < head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) >$ [Definition of $head'$]
$= \frown/ \bullet ([\![!t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of $*$]
$= \frown/ \bullet\ head'(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of !]
$= \frown/ \bullet < head(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) >$ [Definition of $head'$]
$= < head(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) >$ [Definition of $\frown/$]
$= < head(< head(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) >) >$ [Property of $head$]
$= head'(< head(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) >)$ [Definition of $head'$]
$= head'(\frown/ \bullet < head(([\![t_2]\!]\Gamma_L\ \Gamma_T) head([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)) >)$ [Definition of $\frown/$]

$$
\begin{aligned}
&= head'(\frown/ \bullet\ head'((\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T) head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r))) && \text{[Definition of } head'\text{]}\\
&= head'(\frown/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T) head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r)) && \text{[Definition of !]}\\
&= head'(\frown/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet < head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) >) && \text{[Definition of *]}\\
&= head'(\frown/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet\ head'((\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r)) && \text{[Definition of } head'\text{]}\\
&= head'(\frown/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet (\llbracket !t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of !]}\\
&= head'(\llbracket (!t_1;\ !t_2) \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of ;]}\\
&= \llbracket !(!t_1;\ !t_2) \rrbracket \Gamma_L\ \Gamma_T\ r && \text{[Definition of !]}
\end{aligned}
$$

$\square$

Law 13. $!(t_1;\ t_2) = !(t_1;\ !t_2)$

Proof

$$
\begin{aligned}
&\llbracket !(t_1;\ t_2) \rrbracket \Gamma_L\ \Gamma_T\ r\\
&= head'(\llbracket t_1;\ t_2 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of !]}\\
&= head'(\frown/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of ;]}\\
&= < head(\frown/ \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) > && \text{[Definition of } head'\text{]}\\
&= < head(< head((\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) >) > && \text{[Definition of } \frown/\text{]}\\
&= head'(< head((\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) >) && \text{[Definition of } head'\text{]}\\
&= head'(\frown/ \bullet < head((\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) >) && \text{[Definition of } \frown/\text{]}\\
&= head'(\frown/ \bullet\ head' \bullet (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of } head'\text{]}\\
&= head'(\frown/ \bullet (\llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T)* \bullet \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of !]}\\
&= head'(\llbracket t_1;\ !t_2 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of ;]}\\
&= \llbracket !(t_1;\ !t_2) \rrbracket \Gamma_L\ \Gamma_T\ r && \text{[Definition of !]}
\end{aligned}
$$

$\square$

Law 14.(a) $!(t_1 \mid t_2) = !(!t_1 \mid t_2)$

Proof

$$
\begin{aligned}
&\llbracket !(t_1 \mid t_2) \rrbracket \Gamma_L\ \Gamma_T\ r\\
&= head'(\llbracket t_1 \mid t_2 \rrbracket \Gamma_L\ \Gamma_T\ r) && \text{[Definition of !]}\\
&= head'(\frown/ \bullet < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T >^{\circ}\ \ r) && \text{[Definition of !]}\\
&= head'(\frown/ < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r >) && \text{[Definition of } \circ\text{]}\\
&= head'((\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) \frown (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)) && \text{[Definition of } \frown/\text{]}\\
&= < head((\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) \frown (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)) > && \text{[Definition of } head'\text{]}\\
&= < head(< head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) > \frown (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)) > && \text{[Property of } head \text{ with } \frown\text{]}\\
&= head'(< head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) > \frown (\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r)) && \text{[Definition of } head'\text{]}\\
&= head'(\frown/ << head(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ r) >, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r >) && \text{[Definition of } \frown/\text{]}\\
&= head'(\frown/ < head'(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T)\ r, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ r >) && \text{[Definition of } head'\text{]}
\end{aligned}
$$

$= head'(\frown/ < [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >)$ [Definition of !]
$= head'(\frown/ \bullet\!< [\![!t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T >^\circ\ r)$ [Definition of $\circ$]
$= head'([\![!t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= [\![!(!t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T$ [Definition of !]

$\square$

Law 14.(b) $!(t_1 \mid t_2) = !(t_1 \mid !t_2)$

Proof

$[\![!(t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T\ r$
$= head'([\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= head'(\frown/ \bullet\!< [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T >^\circ\ r)$ [Definition of !]
$= head'(\frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of $\circ$]
$= head'(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\frown/$]
$= < head(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r) \frown ([\![t_2]\!]\Gamma_L\ \Gamma_T\ r)) >$ [Definition of $head'$]
$= < head(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)^\frown < head([\![t_2]\!]\Gamma_L\ \Gamma_T\ r) >) >$ [Property of $head$ with $\frown$]
$= head'(([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)^\frown < head([\![t_2]\!]\Gamma_L\ \Gamma_T\ r) >)$ [Definition of $head'$]
$= head'(\frown/ < ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r), < head([\![t_2]\!]\Gamma_L\ \Gamma_T\ r) >>)$ [Definition of $\frown/$]
$= head'(\frown/ < ([\![t_1]\!]\Gamma_L\ \Gamma_T\ r), head'([\![t_2]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $head'$]
$= head'(\frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![!t_2]\!]\Gamma_L\ \Gamma_T\ r >)$ [Definition of !]
$= head'(\frown/ \bullet\!< ([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![!t_2]\!]\Gamma_L\ \Gamma_T) >^\circ\ r)$ [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 !]

$\square$

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

Proof

$[\![!(t_1 \mid t_1;\ t_2)]\!]\Gamma_L\ \Gamma_T\ r$
$= head'([\![t_1 \mid t_1;\ t_2]\!]\Gamma_L\ \Gamma_T\ r)$ [Definition of !]
$= head'(\frown/ \bullet\!< [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_1;\ t_2]\!]\Gamma_L\ \Gamma_T >^\circ\ r)$ [Definition of |]
$= head'(\frown/ \bullet\!< [\![t_1]\!]\Gamma_L\ \Gamma_T, \frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![t_1]\!]\Gamma_L\ \Gamma_T) >^\circ\ r)$
[Definition of ;]
$= head'(\frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, \frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r >)$ [Definition of $\circ$]
[Functional Composition]
$= head'(([\![t_1]\!]\Gamma_L\ \Gamma_T)\ r \frown (\frown/ \bullet ([\![t_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![t_1]\!]\Gamma_L\ \Gamma_T\ r))$ [Definition of $\frown/$]
$= head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r)$ [Lemma D.5]
$= [\![!t_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of !]

$\square$

Law 16. $!(t_1 \mid t_2 \mid t_1) = !(t_1 \mid t_2)$

Proof

$$
\begin{array}{ll}
[\![!(t_1 \mid t_2 \mid t_1)]\!]\Gamma_L\ \Gamma_T\ r & \\
= head'([\![(t_1 \mid t_2 \mid t_1)]\!]\Gamma_L\ \Gamma_T)\ r & [\text{Definition of } !] \\
= head'(^\frown/ \bullet < [\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T, [\![t_1]\!]\Gamma_L\ \Gamma_T >^\circ\ r) & [\text{Definition of } |] \\
= head'(^\frown/ < (^\frown/ \bullet < [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T >^\circ), ([\![t_1]\!]\Gamma_L\ \Gamma_T) >^\circ\ r) & \\
 & [\text{Definition of } |] \\
= head'(^\frown/ < (^\frown/ < ([\![t_1]\!]\Gamma_L\ \Gamma_T), ([\![t_2]\!]\Gamma_L\ \Gamma_T) >^\circ)\ r, [\![t_1]\!]\Gamma_L\ \Gamma_T\ r >) & \\
 & [\text{Definition of } \circ] \\
= head'(^\frown/ < (^\frown/ \bullet < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >), [\![t_1]\!]\Gamma_L\ \Gamma_T\ r >) & \\
 & [\text{Definition of } \circ] \\
= head'(^\frown/ \bullet < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r\ ^\frown\ [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >, [\![t_1]\!]\Gamma_L\ \Gamma_T\ r >) & [\text{Definition of } ^\frown/] \\
= head'([\![t_1]\!]\Gamma_L\ \Gamma_T\ r\ ^\frown\ [\![t_2]\!]\Gamma_L\ \Gamma_T\ r\ ^\frown\ [\![t_1]\!]\Gamma_L\ \Gamma_T\ r) & [\text{Definition of } ^\frown/] \\
= head'(^\frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T\ r, [\![t_2]\!]\Gamma_L\ \Gamma_T\ r >) & [\text{Lemma D.6}] \\
= head'(^\frown/ < [\![t_1]\!]\Gamma_L\ \Gamma_T, [\![t_2]\!]\Gamma_L\ \Gamma_T >^\circ\ r) & [\text{Definition of } \circ] \\
= head'([\![t_1 \mid t_2]\!]\Gamma_L\ \Gamma_T\ r) & [\text{Definition of } |] \\
= [\![!(t_1 \mid t_2)]\!]\Gamma_L\ \Gamma_T\ r & [\text{Definition of } !]
\end{array}
$$

□

Law 17. $!(\mathbf{skip} \mid t) = \mathbf{skip}$

Proof

$$
\begin{array}{ll}
[\![!(\mathbf{skip} \mid t)]\!]\Gamma_L\ \Gamma_T\ r & \\
= head'([\![(\mathbf{skip} \mid t)]\!]\Gamma_L\ \Gamma_T\ r) & [\text{Definition of } !] \\
= head'(^\frown/ \bullet < [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T, [\![t]\!]\Gamma_L\ \Gamma_T >^\circ\ r) & [\text{Definition of } |] \\
= head'(^\frown/ < [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r, [\![t]\!]\Gamma_L\ \Gamma_T\ r >) & [\text{Definition of } \circ] \\
 & [\text{Functional Composition}] \\
= head'(^\frown/ << r >, [\![t]\!]\Gamma_L\ \Gamma_T\ r >) & [\text{Definition of } \mathbf{skip}] \\
= head'(< r >\ ^\frown[\![t]\!]\Gamma_L\ \Gamma_T\ r >) & [\text{Definition of } ^\frown/] \\
= < head(< r >\ ^\frown[\![t]\!]\Gamma_L\ \Gamma_T\ r >) > & [\text{Definition of } head'] \\
= < r > & [\text{Definition of } head] \\
= [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r & [\text{Definition of } \mathbf{skip}]
\end{array}
$$

□

Law 18. $!(t \mid t) = !t$

Proof

$[\![!(t \mid t)]\!] \Gamma_L\ \Gamma_T\ r$	
$= head'([\![(t \mid t)]\!] \Gamma_L\ \Gamma_T\ r)$	[Definition of !]
$= head'(\frown / \bullet < [\![t]\!] \Gamma_L\ \Gamma_T, [\![t]\!] \Gamma_L\ \Gamma_T >^{\circ}\ r)$	[Definition of |]
$= head'(\frown / < [\![t]\!] \Gamma_L\ \Gamma_T\ r, [\![t]\!] \Gamma_L\ \Gamma_T\ r >)$	[Definition of $\circ$]
$= head'([\![t]\!] \Gamma_L\ \Gamma_T\ r \frown [\![t]\!] \Gamma_L\ \Gamma_T\ r)$	[Definition of $\frown /$]
$= < head([\![t]\!] \Gamma_L\ \Gamma_T\ r \frown [\![t]\!] \Gamma_L\ \Gamma_T\ r) >$	[Definition of $head'$]
$= < head([\![t]\!] \Gamma_L\ \Gamma_T\ r) >$	[Property of $head$]
$= head'([\![t]\!] \Gamma_L\ \Gamma_T\ r)$	[Definition of $head'$]
$= [\![!t]\!] \Gamma_L\ \Gamma_T\ r$	[Definition of !]

$\square$

Law 19. $!!t = !t$

$!!t\ r = !t\ r$	
= head'(head'(t r))	[Definition of !]
= head'(t r)	[Lemma D.14]
= !t r	[Definition of !]

$\square$

C.3 Laws involving *succs* and *fails*

Law 20. $succs\ t;\ t = t$

$succs\ t;\ t\ r$	
$= \frown \bullet\ t* \bullet succs\ t\ r$	[Definition of ;]
Case $t\ r = <> = \frown / \bullet\ t* \bullet \mathbf{fail}\ r$	[Definition of $succs$]
$= \frown / \bullet\ t*\ <>$	[Definition of **fail**]
$= \frown /\ <>$	[Definition of $*$]
$= <>$	[Definition of $\frown /$]
$= t\ r$	[Assumption]
Case $t\ r \neq <> = \frown / \bullet\ t* \bullet \mathbf{skip}\ r$	[Definition of $succs$]
$= \frown / \bullet\ t*\ < r >$	[Definition of **skip**]

$= \frown/ < t\ r >$ [Definition of $*$]

$= t\ r$ [Definition of $\frown/$]

$\square$

Law 21. *fails* $t;\ t =$ **fail**

fails $t;\ t\ r$

$= \frown/ \bullet t* \bullet \mathit{fails}\ t\ r$ [Definition of ;]

Case $t\ r =<> = \frown/ \bullet t* \bullet \mathbf{skip}\ \ r$ [Definition of *succs*]

$= \frown/ \bullet t*\ \ < r >$ [Definition of **skip**]

$= \frown/ \bullet < t\ r >$ [Definition of $*$]

$= t\ r$ [Definition of $\frown/$]

$= <>$ [Assumption]

$=$ **fail** r [Definition of **fail**]

Case $t\ r \neq <> = \frown/ \bullet t* \bullet \mathbf{fail}\ \ r$ [Definition of *fails*]

$= \frown/ <>$ [Definition of $*$]

$=$ **fail** r [Definition of $\frown/$]

$\square$

Law 22. $\mathit{fails}\ t = \mathit{fails}\ !t = \mathit{fails}\ !t$

1. $\mathit{fails}\ t = \mathit{fails}\ !t$

$\mathit{fails}\ t\ r$

Case $t\ r =<>$

$=$ **skip** r [Definition of *fails*]

$= \mathit{fails}(\mathit{head}'(t\ r))$ [Definition of $\mathit{fails}, \mathit{head}'$]

[Assumption]

$= \mathit{fails}(!t\ r)$ [Definition of !]

Case $t\ r \neq <>$

$=$ **fail** r [Definition of *fails*]

$= \mathit{fails}(\mathit{head}'(t\ r))$ [Definition of $\mathit{fails}, \mathit{head}'$]

[Assumption]

$= \mathit{fails}(!t\ r)$ [Definition of !]

2. $\mathit{fails}\ !t =!\mathit{fails}\ t\ r$

$\mathit{fails}\ !t$

Case $t\ r = <>$	
$= (\mathbf{skip} \lhd !t\ r = <> \rhd \mathbf{fail})\ r$	[Definition of fails]
$= (\mathbf{skip} \lhd head'(t\ r) = <> \rhd \mathbf{fail})\ r$	[Definition of !]
$= (\mathbf{skip} \lhd head'(<>) = <> \rhd \mathbf{fail})\ r$	[Assumption]
$= \mathbf{skip}\ \ r$	[Definition of head']
$= !\,\mathbf{skip}\ \ r$	[Law 8]
$= !(fails\ t\ r)$	[Definition of fails]
	[Assumption]

Case $t\ r = lst \neq <>$	
$= (\mathbf{skip} \lhd !t\ r = <> \rhd \mathbf{fail})\ r$	[Definition of fails]
$= (\mathbf{skip} \lhd head'(lst) = <> \rhd \mathbf{fail})\ r$	[Definition of !]
	[Assumption]
$= \mathbf{fail}\ \ r$	[Definition of $\lhd\ \rhd$]
$= !\,\mathbf{fail}\ \ r$	[Law 9]
$= !(fails\ t\ r)$	[Definition of fails]
	[Assumption]

$\square$

Law 23. $fails(succs\ t) = fails\ t$

$fails(succs\ t)\ r$

Case $t\ r = <>$	
$fails(succs\ t)\ r$	
$= (\mathbf{skip} \lhd succs\ t\ r = <> \rhd \mathbf{fail})\ r$	[Definition of fails]
$= (\mathbf{skip} \lhd \mathbf{fail}\ \ r = <> \rhd \mathbf{fail})\ r$	[Definition of *succs*]
	[Assumption]
$= (\mathbf{skip} \lhd <> = <> \rhd \mathbf{fail})\ r$	[Definition of **fail**]
$= \mathbf{skip}\ \ r$	[Definition of $\lhd\ \rhd$]
$= fails\ t\ r$	[Definition of *fails*]
	[Assumption]

Case $t\ r \neq <> = (\mathbf{skip} \lhd succs\ t\ r = <> \rhd \mathbf{fail})\ r$	[Definition of fails]
$= (\mathbf{skip} \lhd \mathbf{skip}\ \ r = <> \rhd \mathbf{fail})\ r$	[Definition of *succs*]
	[Assumption]
$= (\mathbf{skip} \lhd < r > = <> \rhd \mathbf{fail})\ r$	[Definition of **skip**]
$= \mathbf{fail}\ \ r$	[Definition of $\lhd\ \rhd$]
$= fails\ t\ r$	[Definition of *fails*]
	[Assumption]

$\square$

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 $t_2\ r = <>$

$= \frown/ \bullet (fails\ t_2)* \bullet fails\ t_1\ r$	[Definition of ;]
$= \frown/ \bullet (fails\ t_2)* \bullet \mathbf{skip}\ \ r$	[Definition of *fails*]
$= \frown/ \bullet (fails\ t_2)* < r >$	[Definition of **skip**]
$= \frown/ < fails\ t_2\ r >$	[Definition of $*$]
$= fails\ t_2\ r$	[Definition of $\frown/$]
$= \mathbf{skip}\ \ r$	[Definition of *fails*]
$= fails\ t_1\ r$	[Definition of *fails*]
$= \frown/ < fails\ t_1\ r >$	[Definition of $\frown/$]
$= \frown/(fails\ t_1)* < r >$	[Definition of $*$]
$= \frown/(fails\ t_1)* \bullet \mathbf{skip}\ \ r$	[Definition of **skip**]
$= \frown/(fails\ t_1)* \bullet fails\ t_2\ r$	[Definition of *fails*]
$= fails\ t_2;\ fails\ t_1\ r$	[Definition of ;]

Case $t_1\ r = <>$ and $t_2\ r \neq <>$

$= \frown/ \bullet (fails\ t_2)* \bullet fails\ t_1\ r$	[Definition of ;]
$= \frown/ \bullet (fails\ t_2)* \bullet \mathbf{skip}\ \ r$	[Definition of *fails*]
$= \frown/ \bullet (fails\ t_2)* < r >$	[Definition of **skip**]
$= \frown/ < fails\ t_2\ r >$	[Definition of $*$]
$= fails\ t_2\ r$	[Definition of $\frown/$]
$= \mathbf{fail}\ \ r$	[Definition of *fails*]
$= <>$	[Definition of *fail*]
$= \frown/ <>$	[Definition of $\frown/$]
$= \frown/(fails\ t_1)* <>$	[Definition of $*$]
$= \frown/(fails\ t_1)* \bullet \mathbf{fail}\ \ r$	[Definition of **fail**]
$= \frown/(fails\ t_1)* \bullet fails\ t_2\ r$	[Definition of *fails*]
$= fails\ t_2;\ fails\ t_1\ r$	[Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r = <>$

$= \frown/ \bullet (fails\ t_2)* \bullet fails\ t_1\ r$	[Definition of ;]
$= \frown/ \bullet (fails\ t_2)* \bullet \mathbf{fail}\ \ r$	[Definition of *fails*]
$= \frown/ \bullet (fails\ t_2)* <>$	[Definition of **fail**]
$= \frown/ <>$	[Definition of $*$]
$= \frown/ <<>>$	[Definition of $\frown/$]
$= \frown/ < \mathbf{fail}\ \ r >$	[Definition of **fail**]
$= \frown/ < fails\ t_1\ r >$	[Definition of *fails*]
$= \frown/(fails\ t_1)*\ \ < r >$	[Definition of $*$]
$= \frown/(fails\ t_1)* \bullet \mathbf{skip}\ \ r$	[Definition of **skip**]
$= \frown/(fails\ t_1)* \bullet fails\ t_2\ r$	[Definition of *fails*]

$= fails\ t_2;\ fails\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r \neq <>$

$= \frown/ \bullet (fails\ t_2)* \bullet fails\ t_1\ r$ [Definition of ;]

$= \frown/ \bullet (fails\ t_2)* \bullet \mathbf{fail}\ \ r$ [Definition of *fails*]

$= \frown/ \bullet (fails\ t_2)* <>$ [Definition of **fail**]

$= \frown/ <>$ [Definition of $*$]

$= <>$ [Definition of $\frown/$]

$= \frown/(fails\ t_1)*\ \ <>$ [Definition of $*, \frown/$]

$= \frown/(fails\ t_1)* \bullet \mathbf{fail}\ \ r$ [Definition of **fail**]

$= \frown/(fails\ t_1)* \bullet fails\ t_2\ r$ [Definition of *fails*]

$= fails\ t_2;\ fails\ t_1\ r$ [Definition of ;]

$\square$

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

$!(t_1 \mid t_2)\ r$

$= head' \bullet \frown/ \bullet < t_1, t_2 >^\circ\ \ r$ [Definition of $!, |$]

$= head' \bullet \frown/ \bullet < t_1\ r, t_2\ r >$ [Definition of $^\circ$]

$= head' \bullet \frown/ < t_1\ r, t_2\ r >$ [Definition of $^\circ$]

$= head'(t_1\ r \frown t_2\ r)$ [Definition of $\frown/$]

Case $t_1\ r = <>$

$= head'(t_2\ r)$ [Property of $head'$]

$= \frown/ <<>, head'(t_2\ r)>$ [Definition of $\frown/$]

$= \frown/ < head'(t_1\ r), \frown/ \bullet (head'\ t_2)*\ \ < r >>$ [Definition of $head', \frown/, *$]

$= \frown/ < head'(t_1\ r), \frown/ \bullet (head'\ t_2) *\ \ \mathbf{skip}\ \ r >$ [Definition of **skip**]

$= \frown/ < head'(t_1\ r), \frown/ \bullet (head'\ t_2) *\ \ fails\ t_1\ r >$ [Definition of *fails*]

$= \frown/ \bullet <!t_1, fails\ t_1;\ !t_2 >^\circ\ \ r$ [Definition of $^\circ, ;\ , !$]

$= !t_1 \mid (fails\ t_1;\ !t_2)\ r$ [Definition of $|$]

Case $t_1\ r \neq <> = head'(t_1\ r \frown <>)$ [Assumption]

$= \frown/ < head'(t_1\ r), <>>$ [Definition of $\frown/$]

$= \frown/ < head'(t_1\ r), \frown/ \bullet (head'\ t_2)* <>) >$ [Definition of $\frown/, *$]

$= \frown/ < head'(t_1\ r), \frown/ \bullet (head'\ t_2)* \bullet failst_1\ r) >$ [Definition of *fails*, **fail**]

$= \frown/ \bullet <!t_1, fails\ t_1;\ !t_2) >^\circ\ \ r$ [Definition of $!, ;\ , ^\circ$]

$= !t_1 \mid (fails\ t_1;\ !t_2)\ r$ [Definition of $|$]

$\square$

Law 26. $!(t_1;\ t_2) = !(t_1;\ succs\ t_2);\ !t_2$

This Law is proved by induction as seen below

$!(t_1;\ t_2)\ r$
$= head' \bullet {}^\frown/ \bullet t_2* \bullet t_1\ r$ [Definition of !, ;]

Base Case: $t\ r = <>$
$= <>$ [Definition of $*, {}^\frown/, head'$]
$= {}^\frown/ \bullet (head'\ t_2)* <>$ [Definition of $*, {}^\frown/$]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'(<>)$ [Definition of $head'$]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip})* <>)$ [Definition of $*, {}^\frown/$]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip})* \bullet\ t_1\ r)$
[Assumption]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (succs\ t_2)* \bullet\ t_1\ r)$ [Definition of $succs$]
$= !(t_1;\ succs\ t_2);\ !t_2$ [Definition of !, ;]

Inductive step: We assume that
$head' \bullet {}^\frown/ \bullet t_2 * ls = {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} \rhd \mathbf{skip}) *\ ls)$
We must prove that
$head' \bullet {}^\frown/ \bullet t_2 * l : ls = {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} \rhd \mathbf{skip}) *\ l : ls)$

$head' \bullet {}^\frown/ \bullet t_2 * l : ls = head' \bullet {}^\frown/(t_2\ l : t_2 *\ ls)$ [Definition of $*$]
$= head'(t_2\ l {}^\frown ({}^\frown/ \bullet t_2 *\ ls))$ [Definition of ${}^\frown/$]
$= t_2\ l$ [Definition of $head'$]

Case $t_2\ l = <>$
$= <>$ [Assumption]
$= {}^\frown/ \bullet (head'\ t_2)* <<>>$ [Definition of $*, {}^\frown/, head'$]
$= {}^\frown/ \bullet (head'\ t_2)* < \mathbf{fail}\ \ l >$ [Definition of **fail**]
$= {}^\frown/ \bullet (head'\ t_2)* < (\mathbf{fail} \lhd t_2\ l = <> \rhd \mathbf{skip})\ l >$ [Definition of $\lhd\ \rhd$, **fail**]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'((\mathbf{fail} \lhd t_2\ l = <> \rhd \mathbf{skip})\ l {}^\frown$
$({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip}) *\ ls))$
[Definition of $head'$]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip}) * l : ls)$ [Definition of $*$]
$= {}^\frown/ \bullet (head'\ t_2)* \bullet head'({}^\frown/ \bullet (succs\ t_2)* \bullet t_1\ r)$ [Definition of $succs$]
$= !(t_1;\ succs\ t_2);\ !t_2$ [Definition of ; , !]

Case $t_2\ l \neq <>$
$= {}^\frown/ \bullet (head'\ t_2)* << l >>$ [Definition of $*, {}^\frown/, head'$]
$= {}^\frown/ \bullet (head'\ t_2)* < \mathbf{skip}\ \ l >$ [Definition of **skip**]
$= {}^\frown/ \bullet (head'\ t_2)* < (\mathbf{fail} \lhd t_2\ l = <> \rhd \mathbf{skip})\ l >$ [Definition of $\lhd\ \rhd$]

$= \frown / \bullet (head'\ t_2)* \bullet head'((\mathbf{fail} \lhd t_2\ l = <> \rhd \mathbf{skip})\ l \frown$
$(\frown / \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip}) *\ ls))$ [Definition of $head'$]
$= \frown / \bullet (head'\ t_2)* \bullet head'(\frown / \bullet (\mathbf{fail} \lhd t_{2\_} = <> \rhd \mathbf{skip}) * l : ls)$ [Definition of $*$]
$= \frown / \bullet (head'\ t_2)* \bullet head'(\frown / \bullet (succs\ t_2)* \bullet t_1\ r)$ [Definition of $succs$]
$= !(t_1;\ succs\ t_2);\ !t_2$ [Definition of ; ,!]

$\square$

Law 27. $succs(t_1 \mid t_2) = !(succs\ t_1 \mid succs\ t_2)$

$succs(t_1 \mid t_2)\ r$

Case $t_1\ r = <>$ and $t_2\ r = <>$
$= \mathbf{fail}\ r$ [Definition of $succs$]
$= <>$ [Definition of **fail**]
$= head'(<>)$ [Definition of $head'$]
$= head'(\frown /\ <>)$ [Definition of $\frown /$]
$= head'(\frown /\ <<>, <>>)$ [Definition of $\frown /$]
$= head'(\frown /\ < \mathbf{fail}\ r, \mathbf{fail}\ r >)$ [Definition of **fail**]
$= head'(\frown /\ < (\mathbf{fail}\ \ dres\ t_1\ r = <> \rhd \mathbf{skip})\ r,$
$(\mathbf{fail}\ \ dres\ t_2\ r = <> \rhd \mathbf{skip})\ r >)$ [Definition of $\lhd\ \rhd$]
[Assumption]
$= head'(\frown / \bullet < (\mathbf{fail}\ \ dres\ t_{1\_} = <> \rhd \mathbf{skip}),$
$(\mathbf{fail}\ \ dres\ t_{2\_} = <> \rhd \mathbf{skip})\ r >^\circ\ r$ [Definition of $\circ$]
$= !(succs\ t_1 \mid succs\ t_2)$ [Definition of $succs, \mid, head'$]

Other Cases
$= \mathbf{skip}\ r$ [Definition of $succs$]
$= < r >$ [Definition of **skip**]
$= head'(< r >)$ [Definition of $head'$]

Case $t_1\ r \neq <>$ and $t_2\ r = <>$
$= head'(\frown / << r >, <>>)$ [Definition of $\frown /$]
$= head'(\frown / < \mathbf{skip}\ r, \mathbf{fail}\ r >)$ [Definition of **fail**, **skip**]

Case $t_1\ r = <>$ and $t_2\ r \neq <>$
$= head'(\frown / <<>, < r >>)$ [Definition of $\frown /$]
$= head'(\frown / < \mathbf{fail}\ r, \mathbf{skip}\ r >)$ [Definition of **fail**, **skip**]

Case $t_1\ r \neq <>$ and $t_2\ r \neq <>$
$= head'(\frown / << r >, < r >>)$ [Definition of $\frown /$]

$= head'(\frown/ < \textbf{skip}\ \ r, \textbf{skip}\ \ r >)$ [Definition of **fail**, **skip**]

In the three cases above, we have that

$= head'(\frown/\ \ < (\textbf{fail}\ \ dres\ t_1\ r =<> \rhd \textbf{skip})\ r,$
$(\textbf{fail}\ \ dres\ t_2\ r =<> \rhd \textbf{skip})\ r >)$ [Definition of $\lhd\ \rhd$]
[Assumption]

$= head'(\frown/ \bullet< (\textbf{fail}\ \ dres\ t_{1-} =<> \rhd \textbf{skip}),$
$(\textbf{fail}\ \ dres\ t_{2-} =<> \rhd \textbf{skip})\ r >^{\circ}\ \ r)$ [Definition of $\circ$]
$= !(succs\ t_1 \mid succs\ t_2)$ [Definition of $succs$, $|$, $head'$]

□

Law 28. $fails(t_1 \mid t_2) = fails\ t_1;\ fails\ t_2$

$fails(t_1 \mid t_2)\ r$

Case $t_1\ r =<>$ and $t_2\ r =<>$
$= \textbf{fail}\ \ r$ [Assumption]
$= <>$ [Definition of **fail**]
$= \frown/ <>$ [Definition of $\frown/$]
$= \frown/ \bullet \textbf{fail} * <>$ [Definition of $*$]
$= \frown/ \bullet \textbf{fail} * \bullet \textbf{fail}\ r$ [Definition of **fail**]
$= \frown/ \bullet (\textbf{skip} \lhd t_{2-} =<> \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_{1-} =<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= fails\ t_1;\ failst_2\ r$ [Definition of $fails$]

Case $t_1\ r \neq<>$ and $t_2\ r =<>$
$= \textbf{skip}\ \ r$ [Assumption]
$= < r >$ [Definition of **skip**]
$= \frown/ < r >$ [Definition of $\frown/$]
$= \frown/ \bullet \textbf{skip} * <>$ [Definition of $*$]
$= \frown/ \bullet \textbf{skip} * \bullet \textbf{fail}\ r$ [Definition of **fail**]
$= \frown/ \bullet (\textbf{skip} \lhd t_{2-} =<> \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_{1-} =<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= fails\ t_1;\ failst_2\ r$ [Definition of $fails$]

Case $t_1\ r =<>$ and $t_2\ r \neq<>$
$= \textbf{fail}\ \ r$ [Assumption]
$= <>$ [Definition of **fail**]
$= \frown/ <>$ [Definition of $\frown/$]
$= \frown/ \bullet \textbf{fail} * < r >$ [Definition of $*$]
$= \frown/ \bullet \textbf{fail} * \bullet \textbf{skip}\ r$ [Definition of **skip**]
$= \frown/ \bullet (\textbf{skip} \lhd t_{2-} =<> \rhd \textbf{fail}) * \bullet (\textbf{skip} \lhd t_{1-} =<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= fails\ t_1;\ failst_2\ r$ [Definition of $fails$]

Case $t_1\ r \neq <>$ and $t_2\ r \neq <>$

$= \mathbf{skip}\ \ r$	[Assumption]
$= < r >$	[Definition of **skip**]
$= \frown / < r >$	[Definition of $\frown /$]
$= \frown / \bullet \mathbf{skip} * <>$	[Definition of $*$]
$= \frown / \bullet \mathbf{skip} * \bullet \mathbf{skip}\ r$	[Definition of **skip**]
$= \frown / \bullet (\mathbf{skip} \lhd t_{2-} =<> \rhd \mathbf{fail}) * \bullet (\mathbf{skip} \lhd t_{1-} =<> \rhd \mathbf{fail})\ r$	[Definition of $\lhd\ \rhd$]
$= \mathit{fails}\ t_1;\ \mathit{failst}_2\ r$	[Definition of *fails*]

$\square$

Law 29. $\mathit{succs}\ s;\ \mathit{succs}(s;\ t) = \mathit{succs}(s;\ t)$

$\mathit{succs}\ s;\ \mathit{succs}\ (s;\ t)\ r$

$= \frown / \bullet (\mathit{succs}\ (s;\ t)) * \bullet\ \mathit{succs}\ s$	[Definition of $*$]

Case $s\ r =<>$

$= \frown / \bullet (\mathit{succs}\ (s;\ t)) * <>$	[Definition of succs, **fail**]
$= <>$	[Definition of $*, \frown /$]
$= \mathbf{fail}\ \ r$	[Definition of **fail**]
$= (\mathbf{fail} \lhd <>=<> \rhd \mathbf{skip})\ r$	[Definition of $\lhd\ \rhd$]
$= (\mathbf{fail} \lhd t*\ \ <>=<> \rhd \mathbf{skip})\ r$	[Definition of $*$]
$= (\mathbf{fail} \lhd t* \bullet s\ \ r =<> \rhd \mathbf{skip})\ r$	[Assumption]
$= \mathit{succs}\ (s;\ t)\ r$	[Definition of ; , suucs]

Case $s\ r \neq <>$

$= \frown / \bullet (\mathit{succs}\ (s;\ t)) *\ \ < r >$	[Definition of succs, **skip**]
$= \mathit{succs}\ (s;\ t)\ \ < r >$	[Definition of $\frown /, *$]

$\square$

Law 30. $\mathit{fails}\ s = \mathit{fails}\ s;\ \mathit{fails}(s;\ t)$

$\mathit{fails}\ s\ r$

$= (\mathbf{skip} \lhd s\ \ r =<> \rhd \mathbf{fail})\ r$	[Definition of **fail**]

Case $s\ r =<>$

$= \mathbf{skip}\ \ r$	[Definition of $\lhd\ \rhd$]
$= < r >$	[Definition of **skip**]

$= \frown/\ <r>$ [Definition of $\frown/$]
$= \frown/ \bullet \textbf{skip}\ r$ [Definition of **skip**]
$= \frown/ \bullet \textbf{skip}* <r>$ [Definition of $*$]
$= \frown/ \bullet \textbf{skip} * \bullet \textbf{skip}\ r$ [Definition of **skip**]
$= \frown/ \bullet (\textbf{skip} \lhd <>=<> \rhd \textbf{fail})* \bullet (\textbf{skip} \lhd <>=<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= \frown/ \bullet (\textbf{skip} \lhd \frown/ \bullet t* \bullet s\ r =<> \rhd \textbf{fail})* \bullet$
$\quad (\textbf{skip} \lhd s\ r =<> \rhd \textbf{fail})\ r$ [Definition of $\frown/, *$]
[Assumption]
$= \mathit{fails}\ s;\ \mathit{fails}(s;\ t)$ [Definition of *fails*, ;]
Case $s\ r \neq <>$
$= \textbf{fail}\ r$ [Definition of $\lhd\ \rhd$]
$= <>$ [Definition of $<>$]
$= \frown/\ <>$ [Definition of $\frown/$]
Let $f = \textbf{skip}$ or $f = \textbf{fail}$
$= \frown/ \bullet f* <>$ [Definition of **skip**]
$= \frown/ \bullet f* \bullet \textbf{fail}\ r$ [Definition of **fail**]
$= \frown/ \bullet (\textbf{skip} \lhd \frown/ \bullet t* \bullet s\ r =<> \rhd \textbf{fail})* \bullet$
$\quad (\textbf{skip} \lhd \frown/ \bullet s\ r =<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= \frown/(\mathit{fails}\ (s;\ t))* \bullet \mathit{fails}\ s$ [Definition of ; ,**fail**]
$= \mathit{fails}\ s;\ \mathit{fails}(s;\ t)$ [Definition of ;]

$\square$

Law 31. $!s;\ \mathit{fails}\ t = \mathit{fails}(!s;\ t);\ !s$

$!s;\ \mathit{fails}\ t\ r$
$= !s;\ \mathit{fails}\ t\ r$
$= \frown/ \bullet (\mathit{fails}\ t)* \bullet \mathit{head}'(s\ r)$ [Definition of $!, \mathit{head}'$]

Case $s\ r =<>$
$= \frown/ \bullet (\mathit{fails}\ t)* <>$ [Definition of head']
$= <>$ [Definition of $*, \frown/$]
$= \frown/ \bullet (\mathit{head}'(s\ r))$ [Definition of head']
[Assumption]
$= \frown/ \bullet (\mathit{head}'\ s)*\ <r>$ [Definition of $*$]
$= \frown/ \bullet (\mathit{head}'\ s) *\ \textbf{skip}\ r$ [Definition of **skip**]
$= \frown/ \bullet (\mathit{head}'\ s)* \bullet (\textbf{skip} \lhd <>=<> \rhd \textbf{fail})\ r$ [Definition of $\lhd\ \rhd$]
$= \frown/ \bullet (\mathit{head}'\ s)* \bullet (\textbf{skip} \lhd t* \bullet \mathit{head}'(s\ r) =<> \rhd \textbf{fail})\ r$ [Definition of $*, \mathit{head}'$]
$= \frown/ \bullet (\mathit{head}'\ s)* \bullet \mathit{fails}(t* \bullet \mathit{heads})\ r$ [Definition of *fails*]
$= \mathit{fails}(!s;\ t);\ !s$ [Definition ; ,!]
Case $s\ r = x : xs$
$= \frown/ \bullet (\mathit{fails}\ t)* <x>$ [Definition of head']

$= fails\ t\ x$ [Definition of $*, \frown/$]
$= (\mathbf{skip} \lhd t\ x = <> \rhd \mathbf{fail})$ [Definition of $fails$]

Case $t\ x = <>$
$= \mathbf{skip}\ x$ [Definition of $\lhd\ \rhd$]
$= < x >$ [Definition of $\mathbf{skip}$]
$= \frown/ < x >$ [Definition of $\frown/$]
$= \frown/ \bullet (head'\ s)* < r >$ [Definition of $head'$]
$= \frown/ \bullet (head'\ s)* \bullet \mathbf{skip}\ r$ [Definition of $\mathbf{skip}$]
$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{skip} \lhd t\ x = <> \rhd \mathbf{fail})\ r$ [Assumption]
$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{skip} \lhd \frown/ \bullet t*\ < x > = <> \rhd \mathbf{fail})\ r$ [Definition of $*, \frown/$]
$= \frown/ \bullet (head'\ s)* \bullet fails(\frown/ \bullet t*\ head'\ s)\ r$ [Definition of $fails, head'$]
$= fails(!s;\ t);\ !s$ [Definition ; ,!]

Case $t\ x \neq <>$
$= \mathbf{fail}\ x$ [Definition of $\lhd\ \rhd$]
$= <>$ [Definition of $\mathbf{skip}$]
$= \frown/ <>$ [Definition of $\frown/$]
$= \frown/ \bullet (head'\ s)* <>$ [Definition of $head'$]
$= \frown/ \bullet (head'\ s)* \bullet \mathbf{fail}\ r$ [Definition of $\mathbf{fail}$]
$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{skip} \lhd t\ x = <> \rhd \mathbf{fail})\ r$ [Assumption]
$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{skip} \lhd \frown/ \bullet t*\ < x > = <> \rhd \mathbf{fail})\ r$ [Definition of $*, \frown/$]
$= \frown/ \bullet (head'\ s)* \bullet fails(\frown/ \bullet t*\ head'\ s)\ r$ [Definition of $fails, head'$]
$= fails(!s;\ t);\ !s$ [Definition ; ,!]

□

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

$fails(fails(s;\ t))r$

$= (\mathbf{skip} \lhd fails\ s;\ t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $fails$]
$= (\mathbf{skip} \lhd (t* \bullet (\mathbf{skip} \lhd s\ r = <> \rhd \mathbf{fail})\ r = <> \rhd \mathbf{fail}))\ r$ [Definition of $fails$]

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 $*$]
$= fails\ t\ r$ [Definition of $fails$]
$= \frown/ << >, fails\ t\ r >$ [Definition of $\frown/$]
$= \frown/ < succs\ s\ r, fails\ t\ r >$ [Assumption]
$= \frown/ < succs\ s\ r, \frown/ \bullet (fails\ t)*\ \bullet fails\ s\ r >$ [Definition of $*$,Assumption]
$= \frown/ < succs\ s, \frown/ \bullet (fails\ t)*\ \bullet fails\ s >^{\circ}$ [Definition of $^{\circ}$]
$= succs\ s \mid (fails\ s;\ fails\ t)\ r$ [Definition of $\mid$]

Case $s\ r \neq <>$

$= (\mathbf{skip} \lhd t* \bullet fails\ r =<> \rhd \mathbf{fail})\ r$ [Definition of $\lhd\ \rhd$]

$= \mathbf{skip}\ \ r$ [Definition of $\mathbf{fail}, *, \lhd\ \rhd$]

$= < r >$ [Definition of $\mathbf{skip}$]

$= \frown/ << r >, <>>$ [Definition of $\frown/$]

$= \frown/ < \mathbf{skip}\ \ r, \frown/ \bullet (fails\ \ t)* <>>$ [Definition of $\mathbf{skip}, *$]

$= \frown/ < (\mathbf{fail} \lhd s\ \ r =<> \rhd \mathbf{skip})\ r, \frown/ \bullet (fails\ \ t)* \bullet$
$\qquad (\mathbf{skip} \lhd s\ \ r =<> \rhd \mathbf{fail})\ r >$ [Definition of $\lhd\ \rhd$]
[Assumption]

$= \frown/ < succs\ \ s\ \ r, \frown/ \bullet (fails\ \ t)* \bullet fails\ \ s\ \ r >$ [Definition of $fails, succs$]

$= \frown/ < succs\ \ s\ \ r, fails\ \ s;\ fails\ \ t\ \ r >$ [Definition of ;]

$= \frown/ < succs\ \ s, fails\ \ s;\ fails\ \ t >^{\circ}$ [Definition of $^{\circ}$]

$= succs\ \ s \mid (fails\ \ s;\ fails\ \ t)\ \ r$ [Definition of $|$]

□

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

$fails(s;\ fails\ \ t)\ \ r$

$= (\mathbf{skip} \lhd s;\ fails\ \ t\ \ r =<> \rhd \mathbf{fail})\ r$ [Definition of $fails$]

$= (\mathbf{skip} \lhd \frown / \bullet (fails\ \ t)* \bullet s\ \ r =<> \rhd \mathbf{fail})\ r$ [Definition of ;]

Case $s\ r =<>$

$= (\mathbf{skip} \lhd <>=<> \rhd \mathbf{fail})\ r$ [Definition of $*, \frown/$]

$= \mathbf{skip}\ \ r$ [Definition of $\lhd\ \rhd$]

$= < r >$ [Definition of $\mathbf{skip}$]

$= \frown/ << r >, <>>$ [Definition of $\frown/$]

$= \frown/ < fails\ \ s\ \ r, succs\ \ s\ \ r >$ [Definition of $fails, succs$]

$= \frown/ < fails\ \ s\ \ r, (\mathbf{fail} \lhd s\ \ r =<> \rhd \mathbf{skip})\ r >$ [Definition of $succs$]

$= \frown/ < fails\ \ s\ \ r, (\mathbf{fail} \lhd \frown / \bullet t* \bullet s\ \ r =<> \rhd \mathbf{skip})\ r >$ [Definition of $*, \frown/$]

$= \frown/ < fails\ \ s\ \ r, succs\ (\frown/ \bullet t* \bullet s)\ r >$ [Definition of $succs$]

$= \frown/ < fails\ \ s\ \ r, succs\ (s;\ t)\ r >$ [Definition of ;]

$= \frown/ < fails\ \ s, succs\ (s;\ t) >^{\circ}$ [Definition of $^{\circ}$]

$= fails\ \ s \mid succs(s;\ t)\ r$ [Definition of $|$]

Case $s\ r \neq <>$

$= (\mathbf{skip} \lhd \frown / \bullet (fails\ \ t)* \bullet s\ \ r =<> \rhd \mathbf{fail})\ r$

We have that

$= (\mathbf{fail} \lhd \frown / \bullet t* \bullet s\ \ r =<> \rhd \mathbf{skip})\ r$

$= \frown \bullet <<>, (\mathbf{fail} \lhd \frown / \bullet t* \bullet s\ \ r =<> \rhd \mathbf{skip})\ r >$ [Definition of $\frown/$]

$= \frown \bullet < failss\ r, succs\ (s;\ t)\ r >$ [Definition of ; , *succs*]
$= \frown / < fails\ s, succs\ (s;\ t) >^{\circ}$ [Definition of $^{\circ}$]
$= fails\ s \mid succs(s;\ t)\ r$ [Definition of $\mid$]

$\square$

Law 34. $fails\ s;\ succs(s;\ t) = \mathbf{fail}$

$fails\ s;\ succs(s;\ t)\ r$

$= \frown / \bullet (succs(s;\ t))* \bullet fails\ s\ r$

Case $s\ r = <>$
$= \frown / \bullet succs\ (s;\ t)\ r$ [Definition of **fail**, **skip**, $*$]
$= \frown / \bullet succs\ (\frown / \bullet t* \bullet s)\ r$ [Definition of ;]
$= <>$ [Definition of *succs*]
[Assumption]
$= \mathbf{fail}\ r$ [Definition of **fail**]

Case $s\ r \neq <>$
$= \frown / \bullet (succs\ (s;\ t))*\ <>$ [Definition of *fails*, **fail**]
$= <>$ [Definition of $*$]
$= \mathbf{fail}\ r$ [Definition of **fail**]

$\square$

Law 35. $fails(t;\ d) = fails(t;\ succs\ d)$

$fails\ (t;\ d)\ r$

$= (\mathbf{skip} \lhd \frown / \bullet d* \bullet t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of *fails*]

Case $s\ r = <>$
$= (\mathbf{skip} \lhd <> = <> \rhd \mathbf{fail})\ r$ [Definition of $\frown$, $*$]
$= (\mathbf{skip} \lhd \frown / \bullet (succs\ d)* \bullet t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $*$]
[Assumption]
$= fails\ (t;\ succs\ d)\ r$ [Definition of *fails*]

Case $s\ r \neq <>$
$= (\mathbf{skip} \lhd \frown / \bullet (succs\ d)* \bullet t\ r = <> \rhd \mathbf{fail})\ r$ [Lemma D.15]
$= fails\ (t;\ succs\ d)\ r$ [Definition of *fails*]

$\square$

Law 36. $!s;\ succs\ t = succs\ (!s;\ t);\ !s$

$!s;\ succs\ t\ r$

$= \frown/ \bullet (succs\ t)* \bullet head'(s\ r)$ [Definition of ;]

Base Case $s\ r = <>$

$= <>$ [Definition of $head', *, \frown/$]

$= \frown/ \bullet (head'\ s)*\ <>$ [Definition of $head', *, \frown/$]

$= \frown/ \bullet (head'\ s)* \bullet \mathbf{fail}\ \ r <>$ [Definition of **fail**]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd <>=<> \rhd \mathbf{skip})\ r$ [Definition of $\lhd\ \rhd$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t* \bullet head'(s\ r) =<> \rhd \mathbf{skip})\ r$

[Definition of $head', *, \frown/$]

$= succs(!s;\ t);\ !s$ [Definition of ; ,!, $succs$]

Inductive step: We assume that it is true for $s\ r = xs$
We must prove that $s\ r = x : xs$

$= \frown/ \bullet (succs\ t)* < x >$ [Definition of $head'$]

$= \frown/ \bullet succs\ t\ x$ [Definition of $*$]

Case $t\ x = <>$

$= <>$ [Definition of $succs$, **fail**, $\frown/$]

$= \frown/ \bullet (head'\ s)*\ <>$ [Definition of $*, \frown/$]

$= \frown/ \bullet (head'\ s)* \bullet \mathbf{fail}\ \ r$ [Definition of **fail**]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd <>=<> \rhd \mathbf{skip})\ r$ [Definition of $\lhd\ \rhd$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ <>=<> \rhd \mathbf{skip})\ r$ [Definition of $\frown/$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t\ x =<> \rhd \mathbf{skip})\ r$ [Assumption]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t*\ < x >=<> \rhd \mathbf{skip})\ r$ [Definition of $*$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t* \bullet head'(s\ r) =<> \rhd \mathbf{skip})\ r$

[Definition of $head'$]

$= succs(!s;\ t);\ !s$ [Definition of ; ,!, $succs$]

Case $t\ x \neq <>$

$= < x >$ [Definition of $succs$, **skip**, $\frown/$]

$= \frown/ \bullet head'(s\ r)$ [Definition of $head'$]

$= \frown/ \bullet (head'\ s)*\ < r >$ [Definition of $*$]

$= \frown/ \bullet (head'\ s)* \bullet \mathbf{skip}\ \ r$ [Definition of **fail**]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t\ x =<> \rhd \mathbf{skip})\ r$ [Assumption, Definition of $\frown/$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t*\ < x >=<> \rhd \mathbf{skip})\ r$ [Definition of $*$]

$= \frown/ \bullet (head'\ s)* \bullet (\mathbf{fail} \lhd \frown/ \bullet t* \bullet head'(s\ r) =<> \rhd \mathbf{skip})\ r$

[Definition of $head'$]

$= succs(!s;\ t);\ !s$ [Definition of ; ,!, $succs$]

$\square$

Law 37. $fails(fails\ t) = succs\ t$

$fails(fails\ t)\ r$

$= (\mathbf{skip} \lhd fails\ t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $fails$]
$= (\mathbf{skip} \lhd (\mathbf{skip}\ \ t\ r = <> \rhd \mathbf{fail})\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $fails$]

Case $t\ r = <>$
$= (\mathbf{skip} \lhd < r >\ \ r = <> \rhd \mathbf{fail})\ r$ [Definition of $\lhd\ \rhd$, **skip**]
$= \mathbf{fail}\ \ r$ [Definition of $\lhd\ \rhd$]
$= (\mathbf{fail} \lhd t\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $\lhd\ \rhd$]
[Assumption]
$= succs\ t\ r$ [Definition of $succs$]

Case $t\ r \neq <>$
$= (\mathbf{skip} \lhd <>\ \ r = <> \rhd \mathbf{fail})\ r$ [Definition of $\lhd\ \rhd$, **fail**]
$= \mathbf{skip}\ \ r$ [Definition of $\lhd\ \rhd$]
$= (\mathbf{fail} \lhd t\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $\lhd\ \rhd$]
[Assumption]
$= succs\ t\ r$ [Definition of $succs$]

$\square$

Law 38. $fails\ t_1;\ succs\ t_2 = succs\ t_2;\ fails\ t_1$

$fails\ t_1;\ succs\ t_2\ r$

$= \frown/ \bullet (succs\ t_2)* \bullet fails\ t_1\ r$
Case $t_1\ r = <>$ and $t_2\ r = <>$
$= \frown/ \bullet (succs\ t_2)* < r >$ [Definition of $fails$, **skip**]
$= \frown/ \bullet < succs\ t_2\ r >$ [Definition of $*$]
$= \frown/ <<>>$ [Definition of $succs, fails$]
$= <>$ [Definition of $\frown/$]
$= \frown/ <>$ [Definition of $\frown/$]
$= \frown/ \bullet (fails\ t_1)*\ \ <>$ [Definition of $*$]
$= \frown/ \bullet (fails\ t_1)* \bullet succs\ t_2\ r$ [Definition of $fails, succs$]

$= succs\ t_2;\ fails\ t_1\ r$ [Definition of ;]

Case $t_1\ r = <>$ and $t_2\ r \neq <>$

$= \frown / \bullet (succs\ t_2) * < r >$ [Definition of $fails$, **skip**]

$= \frown / \bullet < succs\ t_2\ r >$ [Definition of $*$]

$= \frown / << r >>$ [Definition of $succs$, **skip**]

$= \frown / < fails\ t_1\ r >$ [Definition of $fails$, **skip**]

$= \frown / \bullet (fails\ t_1) * < r >$ [Definition of $*$]

$= \frown / \bullet (fails\ t_1) * \bullet\ succs\ t_2\ r$ [Definition of $succs$, **skip**]

$= succs\ t_2;\ fails\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r = <>$

$= \frown / \bullet (succs\ t_2) * <>$ [Definition of $fails$, **fail**]

$= <>$ [Definition of $*, \frown /$]

$= \frown / \bullet (fails\ t_1) * <>$ [Definition of $*, \frown /$]

$= \frown / \bullet (fails\ t_1) * \bullet\ succs\ t_2\ r$ [Definition of $succs$, **fail**]

$= succs\ t_2;\ fails\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r \neq <>$

$= \frown / \bullet (succs\ t_2) * <>$ [Definition of $fails$, **fail**]

$= <>$ [Definition of $*, \frown /$]

$= \frown / < fails\ t_1\ r >$ [Definition of $fails$, **fail**, $\frown /$]

$= \frown / \bullet (fails\ t_1) * \bullet\ succs\ t_2\ r$ [Definition of $succs$, **skip**]

$= succs\ t_2;\ fails\ t_1\ r$ [Definition of ;]

$\square$

Law 39. $succs\ t_1;\ succs\ t_2 = succs\ t_2;\ succs\ t_1$

$succs\ t_1;\ succs\ t_2\ r$

$= \frown / \bullet (succs\ t_2) * \bullet\ succs\ t_1\ r$

Case $t_1\ r = <>$ and $t_2\ r = <>$

$= \frown / \bullet (succs\ t_2) * <>$ [Definition of $succs$, **fail**]

$= <>$ [Definition of $*, \frown /$]

$= \frown / \bullet (succs\ t_1) * <>$ [Definition of $*, \frown /$]

$= \frown / \bullet (succs\ t_1) * \bullet\ succs\ t_2\ r$ [Definition of $succs$, **fail**]

$= succs\ t_2;\ succs\ t_1\ r$ [Definition of ;]

Case $t_1\ r = <>$ and $t_2\ r \neq <>$

$= \frown / \bullet (succs\ t_2) * <>$ [Definition of $succs$, **fail**]

$= <>$ [Definition of $*, \frown/$]
$= \frown/(succs\ t_1\ r)$ [Definition of $\frown/, succs$]
$= \frown/ \bullet (succs\ t_1)* \ <r>$ [Definition of $*$]
$= \frown/ \bullet (succs\ t_1)* \bullet succs\ t_2\ r$ [Definition of $succs$, **skip**]
$= succs\ t_2;\ succs\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r = <>$
$= \frown/ \bullet (succs\ t_2)* <r>$ [Definition of $succs$, **skip**]
$= \frown/\ <succs\ t_2\ r>$ [Definition of $*$]
$= \frown/\ <<>>$ [Definition of $succs$, **fail**]
$= <>$ [Definition of $\frown/$]
$= \frown/\ <>$ [Definition of $\frown/$]
$= \frown/ \bullet (succs\ t_1)* \ <>$ [Definition of $*$]
$= \frown/ \bullet (succs\ t_1)* \bullet succs\ t_2\ r$ [Definition of $succs$, **fail**]
$= succs\ t_2;\ succs\ t_1\ r$ [Definition of ;]

Case $t_1\ r \neq <>$ and $t_2\ r \neq <>$
$= \frown/ \bullet (succs\ t_2)* <r>$ [Definition of $succs$, **skip**]
$= \frown/\ <succs\ t_2\ r>$ [Definition of $*$]
$= \frown/\ <<r>>$ [Definition of $succs$, **skip**]
$= \frown/\ <succs\ t_1\ r>$ [Definition of $succs$, **skip**]
$= \frown/ \bullet (succs\ t_1)* \ <r>$ [Definition of $*$]
$= \frown/ \bullet (succs\ t_1)* \bullet succs\ t_2\ r$ [Definition of $succs$, **skip**]
$= succs\ t_2;\ succs\ t_1\ r$ [Definition of ;]

□

Law 40. $succs(succs\ t) = succs\ t$

$succs(succs\ t)\ r$
Case $t\ r = <>$
$= (\mathbf{fail} \lhd succs\ t\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $succs$]
$= \mathbf{fail}\ \ r$ [Definition of $succs, \lhd\ \rhd$]
$= succs\ t\ r$ [Definition of $succs$]
[Assumption]

Case $t_1\ r \neq <>$
$= (\mathbf{fail} \lhd succs\ t\ r = <> \rhd \mathbf{skip})\ r$ [Definition of $succs$]
$= \mathbf{skip}\ \ r$ [Definition of $succs, \lhd\ \rhd$]
$= succs\ t\ r$ [Definition of $succs$]
[Assumption]

□

Law 41. $succs(fails\ t) = fails\ t$

$succs(fails\ t)\ r$
$= (\textbf{fail} \lhd fails\ t\ r = <> \rhd \textbf{skip})\ r$ [Definition of $succs$]

Case $t\ r = <>$
$= \textbf{skip}\ \ r$ [Definition of $fails, \lhd\ \rhd$]
$= fails\ t\ r$ [Definition of $fails$]

Case $t_1\ r \neq <>$
$= \textbf{fail}\ \ r$ [Definition of $fails, \lhd\ \rhd$]
$= fails\ t\ r$ [Definition of $fails$]

$\square$

Law 42. $succs(t;\ d) = succs(t;\ succs\ d)$

$succs(t;\ d)\ r$
$= (\textbf{fail} \lhd \frown / \bullet\ d* \bullet\ t\ r = <> \rhd \textbf{skip})\ r$ [Definition of $succs, ;$]

Case $t\ r = <>$
$= (\textbf{fail} \lhd \frown / \bullet\ d* \bullet\ t\ r = <> \rhd \textbf{skip})\ r$
We can say that this is the same as
$= (\textbf{fail} \lhd \frown / \bullet\ (succs\ d)* \bullet\ t\ r = <> \rhd \textbf{skip})\ r$

Since the existence of a r_n in $t\ r$ such that $d\ r_n\ = <>$ implies that, in the same way, $succs\ d\ r_n = <>$

$= (\textbf{fail} \lhd t;\ succs\ d\ r = <> \rhd \textbf{skip})\ r$ [Definition of ;]
$= succs\ (t;\ succs\ d)\ r$ [Definition of $succs$]

Case $t_1\ r \neq <>$
$= (\textbf{fail} \lhd \frown / \bullet\ d* \bullet\ t\ r = <> \rhd \textbf{skip})\ r$
We can say that this is the same as
$= (\textbf{fail} \lhd \frown / \bullet\ (succs\ d)* \bullet\ t\ r \neq <> \rhd \textbf{skip})\ r$

Since the existence of a r_n in $t\ r$ such that $d\ r_n\ \neq <>$ implies that, in the same way, $succs\ d\ r_n \neq <>$

$= (\textbf{fail} \lhd t;\ succs\ d\ r = <> \rhd \textbf{skip})\ r$ [Definition of ;]
$= succs\ (t;\ succs\ d)\ r$ [Definition of $succs$]

$\square$

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 $!t \neq <>$. This finishes this part of the proof.

Now let us prove that $succs\ !t = !succs\ t$

$succs\ !t\ r$
Case $t\ r = <>$
$= <>$ [Definition of $!$, $succs$, **fail**]
$= head'\ <>$ [Definition of $head'$]
$= head'(succs\ t\ r)$ [Definition of $succs$, **fail**]
$= !succs\ t\ r$ [Definition of $!$]

Case $t\ r \neq <>$
$= < r >$ [Definition of $!$, $succs$, **skip**]
$= head'\ < r >$ [Definition of $head'$]
$= head'(succs\ t\ r)$ [Definition of $succs$, **skip**]
$= !succs\ t\ r$ [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
$= (\textbf{fail} \lhd \textbf{fail}\ r = <> \rhd \textbf{skip})\ r$ [Definition of $succs$]
$= (\textbf{fail} \lhd <> = <> \rhd \textbf{skip})\ r$ [Definition of **fail**]
$= \textbf{fail}$ [Definition of $\lhd\ \rhd$]

$\square$

Law 46. $\mathit{fails}\ \ \mathbf{skip} = \mathbf{fail}$

$\mathit{fails}\ \ \mathbf{skip}\ \ r$
$= (\mathbf{skip} \lhd \mathbf{skip}\ \ r =<> \rhd \mathbf{fail})\ r$ [Definition of *fails*]
$= (\mathbf{skip} \lhd < r >=<> \rhd \mathbf{fail})\ r$ [Definition of **skip**]
$= \mathbf{fail}$ [Definition of $\lhd\ \rhd$]

$\square$

Law 47. $\mathit{fails}\ \ \mathbf{fail} = \mathbf{skip}$

$\mathit{fails}\ \ \mathbf{fail}\ \ r$
$= (\mathbf{skip} \lhd \mathbf{fail}\ \ r =<> \rhd \mathbf{fail})\ r$ [Definition of *fails*]
$= (\mathbf{skip} \lhd <>=<> \rhd \mathbf{fail})\ r$ [Definition of **fail**]
$= \mathbf{skip}$ [Definition of $\lhd\ \rhd$]

$\square$

Law 48. $\mathit{fails}\ t;\ \mathit{fails}\ t = \mathit{fails}\ t$

$\mathit{fails}\ t;\ \mathit{fails}\ t\ r$
$= \frown/ \bullet (\mathit{fails}\ t)* \bullet \mathit{fails}\ t\ r$

Case $t\ r =<>$
$= \frown/ \bullet (\mathit{fails}\ t)* < r >$ [Definition of $\mathit{fails}, \mathbf{skip}$]
$= \frown/ < \mathit{fails}\ t\ r >$ [Definition of $*$]
$= < r >$ [Definition of $\mathit{fails}, \mathbf{skip}, \frown/$]
$= \mathit{fails}\ t\ r$ [Definition of $\frown/$]

Case $t\ r \neq<>$
$= \frown/ \bullet (\mathit{fails}\ t)*\ \ <>$ [Definition of $\mathit{fails}, \mathbf{fail}$]
$= <>$ [Definition of $*, \frown/$]
$= \mathit{fails}\ t\ r$ [Definition of $\mathit{fails}, \mathbf{fail}$]

$\square$

Law 49. $succs\ t;\ succs\ t = succs\ t$

$succs\ t;\ succs\ t\ r$

$= \frown/ \bullet (succs\ t)* \bullet succs\ t\ r$

Case $t\ r = <>$

$= \frown/ \bullet (succs\ t)* <>$ [Definition of $succs$, **fail**]

$= <>$ [Definition of $*, \frown/$]

$= succs\ t\ r$ [Definition of $succs$, **fail**]

Case $t\ r \neq <>$

$= \frown/ \bullet (fails\ t)* < r >$ [Definition of $succs$, **skip**]

$= \frown/ \bullet < succ\ t\ r >$ [Definition of $*$]

$= < r >$ [Definition of $succs$, **skip**, $\frown/$]

$= succs\ t\ r$ [Definition of **skip**, $succs$]

$\square$

Law 50. $succs\ t;\ fails\ t = fails\ t;\ succs\ t =$ **fail**

$succs\ t;\ fails\ t\ r$

$= \frown/ \bullet (fails\ t)* \bullet succs\ t\ r$ [Definition of ;]

Case $t\ r = <>$

$= \frown/ \bullet (fails\ t)* <>$ [Definition of $succs$, **fail**]

$= <>$ [Definition of $*, \frown/$]

$=$ **fail** r [Definition of **fail**]

$= <>$

$= \frown/ \bullet <<>>$ [Definition of $\frown/$]

$= \frown/ \bullet < succs\ t\ r >$ [Definition of $succs$, **fail**]

$= \frown/ \bullet (succs\ t)* < r >$ [Definition of $*$]

$= \frown/ \bullet (succs\ t)* \bullet fails\ t\ r$ [Definition of $fails$, **skip**]

$= fails\ t;\ succs\ t\ r$ [Definition of ;]

Case $t\ r \neq <>$

$= \frown/ \bullet (fails\ t)* < r >$ [Definition of $succs$, **skip**]

$= \frown/ < fails\ t\ r >$ [Definition of $*$]

$= \frown/ <<>>$ [Definition of $fails$, **fail**]

$= <>$ [Definition of $*, \frown/$]

$=$ **fail** r [Definition of **fail**]

$= <>$

$= \frown/ \bullet (succs\ t)* <>$ [Definition of $\frown/, *$]

$= \frown / \bullet (succs\ t) * \bullet fails\ t\ r$ [Definition of $fails, \mathbf{fail}$]
$= fails\ t;\ succs\ t\ r$ [Definition of ;]

$\square$

Law 51. $fails\ t \mid succs\ t = succs\ t \mid fails\ t = \mathbf{skip}$

$fails\ t \mid succs\ t\ r$
$= \frown / \bullet < fails\ t\ r, succs\ t\ r >$ [Definition of $\mid, \circ$]

Case $t\ r = <>$
$= \frown / << r >, <>>$ [Definition of $fails, succs$]
[Definition of $\mathbf{skip}, \mathbf{fail}$]
$= \frown / <<>, < r >>$ [Definition of $\frown /$]
$= < r >$ [Definition of $\frown /$]
$= \mathbf{skip}\ r$ [Definition of $\mathbf{skip}$]
$= < r >$ [Definition of $\mathbf{skip}$]
$= \frown / <<>, < r >>$ [Definition of $\frown /$]
$= \frown / < succs\ t\ r, fails\ t\ r >$ [Definition of $succs, fails$]
[Definition of $\mathbf{skip}, \mathbf{fail}$]
$= succs\ t \mid fails\ t\ r$ [Definition of $\circ, \frown /$]

Case $t\ r \neq <>$
$= \frown / <<>, < r >>$ [Definition of $fails, succs$]
[Definition of $\mathbf{skip}, \mathbf{fail}$]
$= \frown / << r >, <>>$ [Definition of $\frown /$]
$= < r >$ [Definition of $\frown /$]
$= \mathbf{skip}\ r$ [Definition of $\mathbf{skip}$]
$= < r >$ [Definition of $\mathbf{skip}$]
$= \frown / << r >, <>>$ [Definition of $\frown /$]
$= \frown / < succs\ t\ r, fails\ t\ r >$ [Definition of $succs, fails$]
[Definition of $\mathbf{skip}, \mathbf{fail}$]
$= succs\ t \mid fails\ t\ r$ [Definition of $\circ, \frown /$]

$\square$

Law 52. $succs(t \mid u) = succs\ t \mid (failst;\ succs\ u)$

$succs(t \mid u)\ r$
$= (\mathbf{fail} \lhd {}^\frown / \bullet\!<t, u>^\circ\ \ r =<> \rhd \mathbf{skip})\ r$ [Definition of $succs, \mid$]
$= (\mathbf{fail} \lhd {}^\frown / \bullet\ t\ r\ {}^\frown\ u\ r =<> \rhd \mathbf{skip})\ r$ [Definition of $^\circ, {}^\frown/$]

Case $t\ r =<>, u\ r =<>$
$= <>$ [Definition of $\lhd\ \rhd, \mathbf{fail}$]
$= {}^\frown/ <<>, <>>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ <<>>>$ [Definition of $succs, \mathbf{fail}, {}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ < succs\ u\ r >>$ [Definition of $succs, \mathbf{fail}$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* < r >>$ [Definition of $*$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* \bullet fails\ t\ r >$ [Definition of $fails, \mathbf{skip}$]
$= succs\ t \mid (failst;\ succs\ u)\ r$ [Definition of $^\circ, \mid$]

Case $t\ r =<>, u\ r \neq<>$
$= < r >$ [Definition of $\lhd\ \rhd, \mathbf{skip}$]
$= {}^\frown/ <<>, < r >>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ << r >>>$ [Definition of $succs, \mathbf{fail}, {}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ < succs\ u\ r >>$ [Definition of $succs, \mathbf{skip}$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* < r >>$ [Definition of $*$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* \bullet fails\ t\ r >$ [Definition of $fails, \mathbf{skip}$]
$= succs\ t \mid (failst;\ succs\ u)\ r$ [Definition of $^\circ, \mid$]

Case $t\ r \neq<>$
$= < r >$ [Definition of $\lhd\ \rhd, \mathbf{skip}$]
$= {}^\frown/ << r >, <>>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* <>>$ [Definition of $succs, \mathbf{skip}, *, {}^\frown/$]
$= {}^\frown/ < succs\ t\ r, {}^\frown/ \bullet (succs\ u)* \bullet fails\ t\ r >$ [Definition of $fails, \mathbf{fail}$]
$= succs\ t \mid (failst;\ succs\ u)\ r$ [Definition of $^\circ, \mid$]

$\square$

Law 53. $succs(fails\ s;\ t)) = fails\ s;\ succs\ t$

$succs(fails\ s;\ t))\ r$
$= (\mathbf{fail} \lhd fails\ s;\ t\ r =<> \rhd \mathbf{skip})\ r$ [Definition of $succs$]
$= (\mathbf{fail} \lhd {}^\frown / \bullet\ t* \bullet fails\ s\ r =<> \rhd \mathbf{skip})\ r$ [Definition of ;]

Case $s\ r =<>, t\ r =<>$

$= (\mathbf{fail} \lhd \frown / \bullet t* < r >=<> \rhd \mathbf{skip})\ r$ [Definition of $\mathit{fails}, \mathbf{skip}$]
$= (\mathbf{fail} \lhd t\ r =<> \rhd \mathbf{skip})\ r$ [Definition of $*, \frown/$]
$= <>$ [Definition of $\lhd\ \rhd, \mathbf{fail}$]
$= \frown/ <<>>$ [Definition of $\frown/$]
$= \frown/ < succs\ t\ r >$ [Definition of $succs, \mathbf{fail}$]
$= \frown/(succs\ t)*\ < r >$ [Definition of $*$]
$= \frown/(succs\ t)* \bullet \mathit{fails}\ s\ r$ [Definition of $\mathit{fails}, \mathbf{skip}$]
$= \mathit{fails}\ s;\ succs\ t\ r$ [Definition of ;]

Case $s\ r =<>, t\ r \neq<>$
$= (\mathbf{fail} \lhd \frown / \bullet t* < r >=<> \rhd \mathbf{skip})\ r$ [Definition of $\mathit{fails}, \mathbf{skip}$]
$= (\mathbf{fail} \lhd t\ r =<> \rhd \mathbf{skip})\ r$ [Definition of $*, \frown/$]
$= < r >$ [Definition of $\lhd\ \rhd, \mathbf{skip}$]
$= \frown/ << r >>$ [Definition of $\frown/$]
$= \frown/ < succs\ t\ r >$ [Definition of $succs, \mathbf{skip}$]
$= \frown/(succs\ t)*\ < r >$ [Definition of $*$]
$= \frown/(succs\ t)* \bullet \mathit{fails}\ s\ r$ [Definition of $\mathit{fails}, \mathbf{skip}$]
$= \mathit{fails}\ s;\ succs\ t\ r$ [Definition of ;]

Case $s\ r \neq<>$
$= (\mathbf{fail} \lhd \frown / \bullet t* <>=<> \rhd \mathbf{skip})\ r$ [Definition of $\mathit{fails}, \mathbf{fail}$]
$= <>$ [Definition of $*, \frown/, \mathbf{fail}$]
$= \frown/(succs\ t)*\ <>$ [Definition of $*$]
$= \frown/(succs\ t)* \bullet \mathit{fails}\ s\ r$ [Definition of $\mathit{fails}, \mathbf{fail}$]
$= \mathit{fails}\ s;\ succs\ t\ r$ [Definition of ;]

$\square$

Law 54. $succs(succs\ s;\ t) = succs\ s;\ succs\ t$

$succs(succs\ s;\ t))\ r$
$= (\mathbf{fail} \lhd succs\ s;\ t\ r =<> \rhd \mathbf{skip})\ r$ [Definition of $succs$]
$= (\mathbf{fail} \lhd \frown / \bullet t* \bullet succs\ s\ r =<> \rhd \mathbf{skip})\ r$ [Definition of ;]

Case $s\ r =<>$
$= (\mathbf{fail} \lhd \frown / \bullet t* <>=<> \rhd \mathbf{skip})\ r$ [Definition of $succs, \mathbf{fail}$]
$= <>$ [Definition of $*, \frown/, \lhd\ \rhd, \mathbf{fail}$]
$= \frown/(succs\ t)*\ <>$ [Definition of $*$]
$= \frown/(succs\ t)* \bullet succs\ s\ r$ [Definition of $succs, \mathbf{fail}$]
$= succs\ s;\ succs\ t\ r$ [Definition of ;]

Case $s\ r \neq <>, t\ r = <>$

$= (\mathbf{fail} \lhd {}^\frown / \bullet t* < r > = <> \rhd \mathbf{skip})\ r$	[Definition of $succs$, **skip**]
$= (\mathbf{fail} \lhd t\ r = <> \rhd \mathbf{skip})\ r$	[Definition of $*, {}^\frown/$]
$= <>$	[Definition of $\lhd\ \rhd$, **fail**]
$= {}^\frown/ <<>>$	[Definition of ${}^\frown/$]
$= {}^\frown/ < succs\ t\ r >$	[Definition of $succs$, **fail**]
$= {}^\frown/(succs\ t)* < r >$	[Definition of $*$]
$= {}^\frown/(succs\ t)* \bullet succs\ s\ r$	[Definition of $succs$, **skip**]
$= succs\ s;\ succs\ t\ r$	[Definition of ;]

Case $s\ r \neq <>, t\ r \neq <>$

$= (\mathbf{fail} \lhd {}^\frown / \bullet t* < r > = <> \rhd \mathbf{skip})\ r$	[Definition of $succs$, **skip**]
$= (\mathbf{fail} \lhd t\ r = <> \rhd \mathbf{skip})\ r$	[Definition of $*, {}^\frown/$]
$= < r >$	[Definition of $\lhd\ \rhd$, **skip**]
$= {}^\frown/ << r >>$	[Definition of ${}^\frown/$]
$= {}^\frown/ < succs\ t\ r >$	[Definition of $succs$, **skip**]
$= {}^\frown/(succs\ t)* < r >$	[Definition of $*$]
$= {}^\frown/(succs\ t)* \bullet succs\ s\ r$	[Definition of $succs$, **skip**]
$= succs\ s;\ succs\ t\ r$	[Definition of ;]

□

Law 55. $succs(s;\ fails\ t) = succs\ s;\ fails(s;\ t)$

$succs(s;\ fails\ t)\ r$

$= (\mathbf{fail} \lhd {}^\frown / \bullet (fails\ t)* \bullet s\ r = <> \rhd \mathbf{skip})\ r$	[Definition of $succs$, ;]

Case $s\ r = <>$

$= (\mathbf{fail} \lhd <> = <> \rhd \mathbf{skip})\ r$	[Definition of $*, {}^\frown/$]
$= <>$	[Definition of $\lhd\ \rhd$, **fail**]
$= {}^\frown/ \bullet (fails\ (s;\ t))* <>$	[Definition of ${}^\frown/, *$]
$= {}^\frown/ \bullet (fails\ (s;\ t))* \bullet succs\ s\ r$	[Definition of $succs$, **fail**]
$= succs\ s;\ fails(s;\ t)\ r$	[Definition of ;]

Case $s\ r \neq <>$

$= (\mathbf{skip} \lhd {}^\frown / \bullet (succs\ t)* \bullet s\ r = <> \rhd \mathbf{fail})\ r$	
$= (\mathbf{skip} \lhd {}^\frown / \bullet t* \bullet s\ r = <> \rhd \mathbf{fail})\ r$	[Law 42]
$= fails\ (s;\ t)\ r$	[Definition of ;]
$= {}^\frown/ \bullet (fails\ (s;\ t))* < r >$	[Definition of $*$]
$= {}^\frown/ \bullet (fails\ (s;\ t))* \bullet succs\ s\ r$	[Definition of $succs$, **skip**]
$= succs\ s;\ fails(s;\ t)\ r$	[Definition of ;]

□

Law 56. $succs(s;\ succs\ t) = succs\ s;\ succs\ (s;\ t)$

$succs(s;\ succs\ t)$
$= succs\ (s;\ t)$ [Law 42]
$= succs\ s;\ succs\ (s;\ t)$ [Law 29]

$\square$

Law 57. $fails\ (succs\ s;\ t) = fails\ s \,\big|\, (succs\ s;\ fails\ t)$

fails (succs s;t) r
$= fails\ (succs\ (succs\ s);\ succs\ t)\ r$ [Law 35]
$= fails\ (succs\ s;\ succs\ t)\ r$ [Law 40]
$= (\mathbf{skip} \lhd {}^\frown/ \bullet (succs\ t)* \bullet succs\ s\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $succs, ;$]

Case $s\ r = <>$
$= (\mathbf{skip} \lhd {}^\frown/ \bullet (succs\ t)* <> = <> \rhd \mathbf{fail})\ r$ [Definition of $succs, \mathbf{fail}$]
$= < r >$ [Definition of $*, {}^\frown/, \lhd\ \rhd, \mathbf{skip}$]
$= {}^\frown/ << r >, <>>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ \bullet (fails\ t)*\ \ <>>$ [Definition of $fails, \mathbf{skip}, {}^\frown/, *$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ \bullet (fails\ t)* \bullet succs\ s\ r >$ [Definition of $succs, \mathbf{fail}$]
$= fails\ s \,\big|\, (succs\ s;\ fails\ t)\ r$ [Definition of $;\ \circ, |$]

Case $s\ r \neq <>, t\ r = <>$
$= (\mathbf{skip} \lhd {}^\frown/ \bullet (succs\ t)* < r > = <> \rhd \mathbf{fail})\ r$ [Definition of $succs, \mathbf{skip}$]
$= (\mathbf{skip} \lhd succs\ t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $*, {}^\frown/$]
$= < r >$ [Definition of $succs, \mathbf{fail}, \mathbf{skip}$]
$= {}^\frown/ <<>, < r >>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ < fails\ t\ r >>$ [Definition of $fails, \mathbf{fail}, \mathbf{skip}$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ \bullet (fails\ t)* < r >>$ [Definition of $*$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ \bullet (fails\ t)* \bullet succs\ s\ r >$ [Definition of $succs, \mathbf{skip}$]
$= fails\ s \,\big|\, (succs\ s;\ fails\ t)\ r$ [Definition of $;\ \circ, |$]

Case $s\ r \neq <>, t\ r \neq <>$
$= (\mathbf{skip} \lhd {}^\frown/ \bullet (succs\ t)* < r > = <> \rhd \mathbf{fail})\ r$ [Definition of $succs, \mathbf{skip}$]
$= (\mathbf{skip} \lhd succs\ t\ r = <> \rhd \mathbf{fail})\ r$ [Definition of $*, {}^\frown/$]
$= <>$ [Definition of $succs, \mathbf{skip}$]
[Definition of $\lhd\ \rhd, \mathbf{fail}$]
$= {}^\frown/ <<>, <>>$ [Definition of ${}^\frown/$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ < fails\ t\ r >>$ [Definition of ${}^\frown/, fails, \mathbf{fail}$]
$= {}^\frown/ < fails\ s\ r, {}^\frown/ \bullet (fails\ t)* < r >>$ [Definition of $*$]

$= \frown/ < fails\ s\ r, \frown/ \bullet (fails\ t)* \bullet succs\ s\ r >$ [Definition of $succs$, **skip**]
$= fails\ s \mid (succs\ s;\ fails\ t)\ r$ [Definition of ; $^\circ$, $\mid$]

$\square$

Law 58. $fails(s;\ succs\ t) = fails\ s \mid (succs\ s;\ fails\ (s;\ t))$

This law is proved by induction, as follows:

$fails(s;\ succs\ t)\ r$
$= (\mathbf{skip} \lhd \frown / \bullet (succs\ t)* \bullet s\ r =<> \rhd \mathbf{fail})\ r$ [Definition of $fails$, ;]

Base Case: $s\ r =<>$
$= (\mathbf{skip} \lhd <>=<> \rhd \mathbf{fail})\ r$ [Definition of $*$, $\frown/$]
$= < r >$ [Definition of $\lhd\ \rhd$, **skip**]
$= \frown/ << r >, <>>$ [Definition of $\frown/$]
$= \frown/ < fails\ s\ r, \frown/ \bullet (fails\ (s;\ t))* <>>$ [Definition of $fails$, **skip**, $*$, $\frown/$]
$= \frown/ < fails\ s\ r, \frown/ \bullet (fails\ (s;\ t))* \bullet succs\ s\ r >$ [Definition of $succs$, **fail**]
$= fails\ s \mid (succs\ s;\ fails\ (s;\ t))\ r$ [Definition of ; ,$^\circ$, $\mid$]

Inductive step: We assume that this law is true for s r = xs
We must prove this law is true for $s\ r = x : xs$

$= (\mathbf{skip} \lhd \frown / \bullet (succs\ t)* \bullet x : xs =<> \rhd \mathbf{fail})\ r$ [Definition of $fails$, ;]
$= (\mathbf{skip} \lhd succs\ t\ x \frown (\frown/ \bullet (succs\ t) *\ xs) =<> \rhd \mathbf{fail})\ r$ [Definition of $*$, $\frown/$]

Case $t\ x =<>$
$= (\mathbf{skip} \lhd \frown / \bullet (succs\ t) *\ xs =<> \rhd \mathbf{fail})\ r$ [Definition of $suucs$, **fail**, $\frown$]
$= \frown/ < fails\ s, \frown/ \bullet (fails\ (\frown/ \bullet t* \bullet s))* \bullet succs\ s >^\circ$ [Assumption]
$= fails\ s \mid (succs\ s;\ fails\ (s;\ t))\ r$ [Definition of ; , $\mid$]

Case $t\ x \neq<>$
$= <>$ [Definition of $\lhd\ \rhd$, **fail**]
$= \frown/ <<>, <>>$ [Definition of $\frown/$]
$= \frown/ < fails\ s\ r, \frown \bullet \mathbf{fail}\ \ r >$ [Assumption]
[Definition of $fails$, **fail**]
$= \frown/ < fails\ s\ r, \frown \bullet (\mathbf{skip} \lhd t\ x \frown \_ =<> \rhd \mathbf{fail})\ r >$ [Definition of $\lhd\ \rhd$]
$= \frown/ < fails\ s\ r, \frown \bullet (\mathbf{skip} \lhd t\ x \frown (\frown/ \bullet t *\ xs) =<> \rhd \mathbf{fail})\ r >$
[Definition of $\frown/, *$]
$= \frown/ < fails\ s\ r, \frown \bullet (\mathbf{skip} \lhd t* \bullet s\ r =<> \rhd \mathbf{fail})\ r >$ [Definition of $\frown/, *$]
$= \frown/ < fails\ s\ r, \frown \bullet (\mathbf{skip} \lhd t* \bullet s\_ =<> \rhd \mathbf{fail})*\ < r >>$ [Definition of $*$]
$= \frown/ < fails\ s\ r, \frown \bullet (\mathbf{skip} \lhd t* \bullet s\_ =<> \rhd \mathbf{fail})* \bullet succs\ s\ r >$

[Definition of $succs$, **skip**]

$= \frown / < fails\ s, \frown \bullet (\mathbf{skip} \lhd t* \bullet s\_ = <> \rhd \mathbf{fail})* \bullet succs\ s >^\circ\ \ r$ [Definition of ; , $fails$, $^\circ$]

$= fails\ s \mid (succs\ s;\ fails\ (s;\ t))\ r$ [Definition of ; , $|$]

$\square$

C.4 Laws involving Structural Combinators

Law 59. $(t_1 \boxed{;} t_2);\ (t_3 \boxed{;} t_4) = (t_1;\ t_3) \boxed{;} (t_2;\ t_4)$

Proof

$$
\begin{array}{ll}
(\llbracket (t_1 \boxed{;} t_2);\ (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)(p_1;\ p_2, proofs) & \\
= \frown / \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) & [\text{Definition of } ;] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* \bullet & \\
\quad \Omega_{sc} * (\prod (< \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)) & [\text{Definition of } \boxed{;}] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* \bullet & \\
\quad \Omega_{sc}* < (p_1', pr_1') : as \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad as \leftarrow \prod < \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >> & [\text{Definition of } \prod] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* \bullet & \\
\quad \Omega_{sc}* < (p_1', pr_1') : as \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad as \leftarrow e2l* < \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >> & [\text{Definition of } \prod] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* \bullet & \\
\quad \Omega_{sc}* << (p_1', pr_1'), (p_2', pr_2') > & \\
\qquad \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > & [\text{Lemma D.7}] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* & \\
\quad < \Omega_{sc} < (p_1', pr_1'), (p_2', pr_2') > & \\
\qquad \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > & [\text{Definition of } *] \\
= \frown / \bullet (\llbracket (t_3 \boxed{;} t_4) \rrbracket \Gamma_L\ \Gamma_T)* & \\
\quad < (p_1';\ p_2', pr_1' \cup pr_2') \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > & [\text{Definition of } \Omega_{sc}] \\
= \frown / < \llbracket t_3 \boxed{;} t_4 \rrbracket \Gamma_L\ \Gamma_T (p_1';\ p_2', pr_1' \cup pr_2') & \\
\qquad \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > & [\text{Definition of } *] \\
= \frown / < \Omega_{sc} * (\prod < \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \llbracket t_4 \rrbracket \Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') >) & \\
\qquad \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > & [\text{Definition of } \boxed{;}] \\
= \frown / < \Omega_{sc}* < (p_3', pr_3') : cs & \\
\qquad \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), & \\
\qquad (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs), & \\
\qquad (p_3', pr_3') \leftarrow \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), & \\
\qquad cs \leftarrow \prod < \llbracket t_4 \rrbracket \Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > & [\text{Definition of } \prod]
\end{array}
$$

$$
\begin{aligned}
= {} & \frown/ < \Omega_{sc}* < (p_3', pr_3') : cs \\
& \qquad\qquad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \qquad\qquad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \qquad\qquad\ \ (p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \\
& \qquad\qquad\ \ cs \leftarrow e2l *\ [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > && [\text{Definition of } \textstyle\prod] \\
= {} & \frown/ < \Omega_{sc}* << (p_3', pr_3'), (p_4', pr_4') > \\
& \qquad\qquad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \qquad\qquad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \qquad\qquad\ \ (p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \\
& \qquad\qquad\ \ (p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > && [\text{Lemma D.7}] \\
= {} & \frown/ < \Omega_{sc} < (p_3', pr_3'), (p_4', pr_4') > \\
& \qquad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \qquad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \qquad\ \ (p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \\
& \qquad\ \ (p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > && [\text{Property of } *] \\
= {} & \frown/ < (p_3';\ p_4', pr_3' \cup pr_4') \\
& \qquad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \qquad (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \qquad\ \ (p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \\
& \qquad\ \ (p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > && [\text{Definition of } \Omega_{sc}] \\
= {} & < (p_3';\ p_4', pr_3' \cup pr_4') \\
& \quad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \quad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \quad\ \ (p_3', pr_3') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1' \cup pr_2'), \\
& \quad\ \ (p_4', pr_4') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_1' \cup pr_2') > && [\text{Property of } \frown/] \\
= {} & < (p_3';\ p_4', pr_3' \cup pr_4') \\
& \quad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \quad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \quad\ \ (p_3', pr_3') \leftarrow (add\ pr_2') * [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'), \\
& \quad\ \ (p_4', pr_4') \leftarrow (add\ pr_1') * [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2') > && [\text{Definition of } add] \\
= {} & < (p_3';\ p_4', pr_3' \cup pr_4') \\
& \quad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \quad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \quad\ \ (p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'), \\
& \quad\ \ (p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2'), \\
& \quad\ \ pr_3' = pr_3'' \cup pr_2', pr_4' = pr_4'' \cup pr_1' > && [\text{Property of } add] \\
= {} & < (p_3';\ p_4', pr_3'' \cup pr_4'' \cup pr_2' \cup pr_1') \\
& \quad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \quad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs), \\
& \quad\ \ (p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'), \\
& \quad\ \ (p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2'), \\
& \quad\ \ pr_3' = pr_3'' \cup pr_2', pr_4' = pr_4'' \cup pr_1' > && [\text{Property of } \cup] \\
= {} & < (p_3';\ p_4', pr_3'' \cup pr_4'') \\
& \quad \mid (p_1', pr_1') \leftarrow [\![t_1]\!]\Gamma_L\ \Gamma_T\ (p_1, proofs), \\
& \quad\ \ (p_2', pr_2') \leftarrow [\![t_2]\!]\Gamma_L\ \Gamma_T\ (p_2, proofs) \\
& \quad\ \ (p_3', pr_3'') \leftarrow [\![t_3]\!]\Gamma_L\ \Gamma_T\ (p_1', pr_1'), \\
& \quad\ \ (p_4', pr_4'') \leftarrow [\![t_4]\!]\Gamma_L\ \Gamma_T\ (p_2', pr_2') > && [\text{Lemma D.8}]
\end{aligned}
$$

$$
\begin{aligned}
= &< \Omega_{sc} < (p_3', pr_3''), (p_4', pr_4'') > \\
&\quad | \ (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\quad\ \ (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) \\
&\quad\ \ (p_3', pr_3'') \leftarrow \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1', pr_1'), \\
&\quad\ \ (p_4', pr_4'') \leftarrow \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2', pr_2') > && [\text{Definition of } \Omega_{sc}] \\
= &\ \Omega_{sc} * << (p_3', pr_3''), (p_4', pr_4'') > \\
&\qquad | \ (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\ \ (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) \\
&\qquad\ \ (p_3', pr_3'') \leftarrow \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1', pr_1'), \\
&\qquad\ \ (p_4', pr_4'') \leftarrow \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2', pr_2') > && [\text{Property of } *] \\
= &\ \Omega_{sc} * (\prod << (p_3', pr_3'') \\
&\qquad\qquad | \ (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\qquad\ \ (p_3', pr_3'') \leftarrow \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1', pr_1') >, \\
&\qquad\quad < (p_4', pr_4'') \\
&\qquad\qquad | \ (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}), \\
&\qquad\qquad\ \ (p_4', pr_4'') \leftarrow \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2', pr_2') >>) && [\text{Definition of } \textstyle\prod] \\
= &\ \Omega_{sc} * (\prod < {}^\frown/ < (p_3', pr_3'') \\
&\qquad\qquad\quad | \ (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\qquad\quad\ \ (p_3', pr_3'') \leftarrow \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1', pr_1') >, \\
&\qquad\quad {}^\frown/ < (p_4', pr_4'') \\
&\qquad\qquad\quad | \ (p_2', pr_2') \leftarrow \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}), \\
&\qquad\qquad\quad\ \ (p_4', pr_4'') \leftarrow \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2', pr_2') >>) && [\text{Property of } {}^\frown/] \\
= &\ \Omega_{sc} * (\prod < {}^\frown/ \bullet \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T * < (p_1', pr_1') \mid (p_1', pr_1') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad {}^\frown/ \bullet \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T * < (p_2', pr_2') \mid (p_2', pr_2') \leftarrow \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >) \\
& && [\text{Definition of } \llbracket\rrbracket \text{ and } *] \\
= &\ \Omega_{sc} * (\prod < {}^\frown/ \bullet \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T * \bullet \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad {}^\frown/ \bullet \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T * \bullet \llbracket t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >) && [\text{Definition of } \llbracket\rrbracket] \\
= &\ \Omega_{sc} * (\prod < \llbracket t_1; \ t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad \llbracket t_2; \ t_4 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >) && [\text{Definition of } ;] \\
= &\ \llbracket (t_1; \ t_3) \boxed{;} (t_2; \ t_4) \rrbracket \Gamma_L \ \Gamma_T \ (p_1; \ p_2, \mathit{proofs}) && [\text{Definition of } \boxed{;}]
\end{aligned}
$$

$\square$

Law 60. $t_1 \boxed{;} (t_2 \mid t_3) = (t_1 \boxed{;} t_2) \mid (t_1 \boxed{;} t_3)$

Proof

$$
\begin{aligned}
&\llbracket t_1 \boxed{;} (t_2 \mid t_3) \rrbracket \Gamma_L \ \Gamma_T \ (p_1; \ p_2, \mathit{proofs}) \\
= &\ \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad \llbracket t_2 \mid t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >) && [\text{Definition of } \boxed{;}] \\
= &\ \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad {}^\frown/ < \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T, \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T >^\circ \ (p_2, \mathit{proofs}) >) && [\text{Definition of } |] \\
= &\ \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \\
&\qquad\quad (\llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs})) {}^\frown (\llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs})) >) && [\text{Definition of } \circ] \\
= &\ \Omega_{sc} * ((\prod < \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \llbracket t_2 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >) {}^\frown \\
&\quad (\prod < \llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p_1, \mathit{proofs}), \llbracket t_3 \rrbracket \Gamma_L \ \Gamma_T \ (p_2, \mathit{proofs}) >)) && [\text{Lemma D.9}]
\end{aligned}
$$

$$
\begin{aligned}
&= \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)^\frown \\
&\quad \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Lemma D.11]} \\
&= {}^\frown/(\Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)^\frown \\
&\qquad \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)) && \text{[Property of } {}^\frown/\text{]} \\
&= {}^\frown/(\llbracket t_1 \boxed{;} t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs)^\frown \\
&\qquad \llbracket t_1 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs)) && \text{[Definition of } \boxed{;}\text{]} \\
&= {}^\frown/ < \llbracket t_1 \boxed{;} t_2 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_1 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T >^\circ\ (p_1;\ p_2, proofs)) && \text{[Definition of } {}^\circ\text{]} \\
&= \llbracket (t_1 \boxed{;} t_2) \mid (t_1 \boxed{;} t_3) \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs)) && \text{[Definition of } \mid \text{]}
\end{aligned}
$$

□

Law 61. $(t_1 \mid t_2) \boxed{;} t_3 = (t_1 \boxed{;} t_3) \mid (t_2 \boxed{;} t_3)$

Proof

$$
\begin{aligned}
&\llbracket (t_1 \mid t_2) \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs) \\
&= \Omega_{sc} * (\prod < \llbracket t_1 \mid t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Definition of } \boxed{;}\text{]} \\
&= \Omega_{sc} * (\prod < {}^\frown/ \bullet < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T >^\circ\ (p_1, proofs), \\
&\qquad\qquad \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Definition of } \mid \text{]} \\
&= \Omega_{sc} * (\prod < {}^\frown/(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs) \\
&\qquad\qquad {}^\frown \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs)), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Definition of } {}^\circ\text{]} \\
&= \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs) \\
&\qquad\qquad {}^\frown \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Property of } {}^\frown/\text{]} \\
&= \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) > {}^\frown \\
&\qquad \prod < \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Lemma D.10]} \\
&= \Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)^\frown \\
&\quad \Omega_{sc} * (\prod < \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Lemma D.11]} \\
&= {}^\frown/(\Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)^\frown \\
&\qquad \Omega_{sc} * (\prod < \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_3 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)) && \text{[Property of } {}^\frown/\text{]} \\
&= {}^\frown/(\llbracket t_1 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T (p_1;\ p_2, proofs)^\frown \llbracket t_2 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T (p_1;\ p_2, proofs)) && \text{[Definition of } \boxed{;}\text{]} \\
&= {}^\frown/ \bullet < \llbracket t_1 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T, \llbracket t_2 \boxed{;} t_3 \rrbracket \Gamma_L\ \Gamma_T >^\circ\ (p_1;\ p_2, proofs)) && \text{[Definition of } {}^\circ\text{]} \\
&= \llbracket (t_1 \boxed{;} t_3) \mid (t_2 \boxed{;} t_3) \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs)) && \text{[Definition of } \mid \text{]}
\end{aligned}
$$

□

Law 62. $!(t_1 \boxed{;} t_2) = !t_1 \boxed{;} !t_2)$

Proof

$$
\begin{aligned}
&\llbracket !(t_1 \boxed{;} t_2) \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs) \\
&= head'(\Omega_{sc} * (\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)) && \text{[Definition of ! and } \boxed{;}\text{]} \\
&= \Omega_{sc} * (head'(\prod < \llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >)) && \text{[Lemma D.12]} \\
&= \Omega_{sc} * (\prod < head'(\llbracket t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs)), \\
&\qquad\qquad head'(\llbracket t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs)) >) && \text{[Lemma D.13]} \\
&= \Omega_{sc} * (\prod < \llbracket !t_1 \rrbracket \Gamma_L\ \Gamma_T\ (p_1, proofs), \llbracket !t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_2, proofs) >) && \text{[Definition of !]} \\
&= \llbracket !t_1 \boxed{;} !t_2 \rrbracket \Gamma_L\ \Gamma_T\ (p_1;\ p_2, proofs) && \text{[Definition of } \boxed{;}\text{]}
\end{aligned}
$$

□

C.5 Laws on *equals*

Law 63. $equals\ r_1;\ equals\ r_2 = fail\ \ if\ \ r_1 \neq r_2$

Proof

$[\![(equals\ r_1;\ equals\ r_2)]\!]\Gamma_L\ \Gamma_T\ r$
$= \frown/ \bullet ([\![equals\ r_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![equals\ r_1]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of ;]

First suppose that $r_1 = r$, we can continue as follows

$= \frown/ \bullet ([\![equals\ r_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of *equals*]
$= \frown/ \bullet ([\![equals\ r_2]\!]\Gamma_L\ \Gamma_T)* < r >$ [Definition of **skip**]
$= \frown/ < [\![equals\ r_2]\!]\Gamma_L\ \Gamma_T\ r >$ [Definition of ∗]
$= \frown/ < [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r >$ [$r_1 = r \wedge r_1 \neq r_2 \Rightarrow r \neq r_2$]
$= \frown/ <<>>$ [Definition of **fail**]
$= <>$ [Definition of $\frown/$]
$= [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of **fail**]

Now suppose that $r_1 \neq r$, we can continue as follows

$= \frown/ \bullet ([\![equals\ r_2]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of *equals*]
$= \frown/ \bullet ([\![equals\ r_2]\!]\Gamma_L\ \Gamma_T)* <>$ [Definition of **fail**]
$= \frown/ <>$ [Definition of ∗]
$= <>$ [Definition of $\frown/$]
$= [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r$ [Definition of **fail**]

□

Law 64. $equals\ r;\ equals\ r = equals\ r$

Proof

$[\![(equals\ r;\ equals\ r)]\!]\Gamma_L\ \Gamma_T\ r_1$
$= \frown/ \bullet ([\![equals\ r]\!]\Gamma_L\ \Gamma_T)* \bullet [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of ;]

First, suppose that $r = r_1$, we can continue as follows

$= \frown/ \bullet ([\![equals\ r]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of *equals*]
$= \frown/ \bullet ([\![equals\ r]\!]\Gamma_L\ \Gamma_T)* < r_1 >$ [Definition of **skip**]
$= \frown/ < [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1 >$ [Definition of ∗]
$= [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of $\frown/$]

Now, suppose that $r \neq r_1$

$= \frown / \bullet ([\![equals\ r]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of *equals*]
$= \frown / \bullet ([\![equals\ r]\!]\Gamma_L\ \Gamma_T)* <>$ [Definition of **fail**]
$= \frown / <>$ [Definition of $*$]
$= <>$ [Definition of $\frown /$]
$= [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r_1$ [*Definitionof* **fail**]
$= [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1$ [Assumption $r \neq r_1$]
[Definition of *equals*]

□

Law 65. $!equals\ r = equals\ r$

Proof

Suppose $r = r_1$, we can continue as follows

$[\![!(equals\ r)]\!]\Gamma_L\ \Gamma_T\ r_1$
$= head'([\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1)$ [Definition of !]
$= head'([\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r_1)$ [Definition of *equals*]
$= head'(< r_1 >)$ [Definition of **skip**]
$= < r_1 >$ [Definition of $head'$]
$= ([\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r_1)$ [Definition of **skip**]
$= [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1$
[Assumption]
[Definition of *equals*]

Now suppose $r \neq r_1$, we can continue as follows

$!(equals\ r\ r_1)$
$= head'([\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1))$ [Definition of !]
$= head'([\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r_1)$ [Assumption]
[Definition of *equals*]
$= head'(<>)$ [Definition of **fail**]
$= <>$ [Definition of $head'$]
$= [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of **fail**]
$= [\![equals\ r]\!]\Gamma_L\ \Gamma_T\ r_1$ [Assumption]
[Definition of *equals*]

□

C.6 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 66. $(\textbf{con}\ \ v \bullet t_1;\ t_2) = (\textbf{con}\ \ v \bullet t_1);\ t_2$ provided $v \notin \phi t_2$

Proof

$([\![\textbf{con}\ \ v \bullet t_1;\ t_2]\!]\Gamma_L\ \Gamma_T)$	
$= ([\![(\vert_{v \in TERM}(t_1;\ t_2)(v))]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]
$= ([\![(\vert_{v \in TERM}(t_1(v);\ t_2(v)))]\!]\Gamma_L\ \Gamma_T)$	[Distribution of ;]
$= ([\![(\vert_{v \in TERM}t_1(v);\ t_2)]\!]\Gamma_L\ \Gamma_T))$	[provided $v \notin \phi t_2$]
$= ([\![(\vert_{v \in TERM}t_1(v));\ t_2]\!]\Gamma_L\ \Gamma_T)$	[Law 6]
$= ([\![(\textbf{con}\ \ v \bullet t_1);\ t_2]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]

□

Law 67. $(\textbf{con}\ \ v \bullet t_1;\ t_2) = t_1;\ (\textbf{con}\ \ v \bullet t_2)$ provided $v \notin \phi t_1$ and $!t_1 = t_1$

Proof

$([\![(\textbf{con}\ \ v \bullet t_1;\ t_2)]\!]\Gamma_L\ \Gamma_T)$	
$= ([\![(\vert_{v \in TERM}(t_1;\ t_2)(v))]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]
$= ([\![(\vert_{v \in TERM}(t_1(v);\ t_2(v)))]\!]\Gamma_L\ \Gamma_T)$	[Distribution of ;]
$= ([\![(\vert_{v \in TERM}t_1;\ t_2(v))]\!]\Gamma_L\ \Gamma_T)$	[provided $v \notin \phi t_2$]
$= ([\![t_1;\ (\vert_{v \in TERM}t_2(v))]\!]\Gamma_L\ \Gamma_T)$	[Law 7]
$= ([\![t_1;\ (\textbf{con}\ \ v \bullet t_2)]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]

□

Law 68. $(\textbf{con}\ \ v \bullet t) = t$ provided $v \notin \phi t$

Proof

$([\![(\textbf{con}\ \ v \bullet t)]\!]\Gamma_L\ \Gamma_T)$	
$= ([\![(\vert_{v \in TERM}t(v))]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]
$= ([\![(\vert_{v \in TERM}t)]\!]\Gamma_L\ \Gamma_T)$	[Provided $v \notin \phi t$]
$= ([\![t]\!]\Gamma_L\ \Gamma_T)$	[$t \mid t = t$]

□

Law 69. $(\mathbf{con}\ \ v \bullet t) = (\mathbf{con}\ \ u \bullet t[v \setminus u])$ provided $u \notin \phi t$

Proof

$([\![(\mathbf{con}\ \ v \bullet t)]\!]\Gamma_L\ \Gamma_T)$	
$= ([\![(\vert_{v \in TERM} t(v))]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]
$= ([\![(\vert_{u \in TERM} t(u))]\!]\Gamma_L\ \Gamma_T)$	[provided $u \notin \phi t$]
$= ([\![(\mathbf{con}\ \ u \bullet t[v \setminus u])]\!]\Gamma_L\ \Gamma_T)$	[Definition of **con**]

$\square$

Law 70. $equals\ r;\ (\mathbf{con}\ \ r' \bullet equals\ r';\ t) = equals\ r;\ t[r' \setminus r])$

$[\![(equals\ r;\ (\mathbf{con}\ \ r' \bullet equals\ r';\ t))]\!]\Gamma_L\ \Gamma_T\ r_1$	
$= [\![(equals\ r;\ (\vert_{r' \in RMCell} equals\ r';\ t(r')))]\!]\Gamma_L\ \Gamma_T\ r_1$	[Definition of **con**]
$= [\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![(equals\ r)]\!]\Gamma_L\ \Gamma_T\ r_1$	[Definition of ;]

First, suppose $r = r_1$. we can continue as follows

$= [\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T)* \bullet ([\![\mathbf{skip}]\!]\Gamma_L\ \Gamma_T\ r_1$	[Assumption]
	[Definition of *equals*]
$= ([\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T)* < r_1 >$	[Definition of **skip**]
$= [\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T r_1$	[Definition of $*$]

One of the choices will take the right RMCell r', this is the one which is equal to r_1. we can continue as follows

$= [\![(equals\ r_1;\ t(r_1))]\!]\Gamma_L\ \Gamma_T r_1$	[Right Choice of Alternation]
$= [\![(equals\ r;\ t(r))]\!]\Gamma_L\ \Gamma_T r_1$	[Assumption]
$= [\![(equals\ r;\ t[r' \setminus r])]\!]\Gamma_L\ \Gamma_T r_1$	[Definition of $\setminus$]

Now, suppose $r \neq r_1$. we can continue as follows

$= ([\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T)* \bullet [\![\mathbf{fail}]\!]\Gamma_L\ \Gamma_T\ r_1$	[Assumption]
	[Definition of *equals*]
$= ([\![(\vert_{r' \in RMCell} equals\ r';\ t(r'))]\!]\Gamma_L\ \Gamma_T)* <>$	[Definition of **fail**]
$= <>$	[Definition of $*$]

By other hand, if we apply

$[\![(equals\ r;\ t[r' \setminus r])]\!]\Gamma_L\ \Gamma_T\ r_1$
$= ([\![(t[r' \setminus r])]\!]\Gamma_L\ \Gamma_T)* \bullet [\![(equals\ r)]\!]\Gamma_L\ \Gamma_T\ r_1$ [Definition of ;]
$= ([\![(t[r' \setminus r])]\!]\Gamma_L\ \Gamma_T)* \bullet [\![(\mathbf{fail})]\!]\Gamma_L\ \Gamma_T\ r_1$ [Assumption]
[Definition of *equals*]
$= ([\![(t[r' \setminus r])]\!]\Gamma_L\ \Gamma_T)* <>$ [Definition of **fail**]
$= <>$ [Definition of $*$]

□

So we get the same result. This ends the proof.

D Lemmas

D.1 Lemma 7

$$f* \bullet \frown/ = \frown/ \bullet f**$$

This Lemma can be proved by induction as follows:

Base Case : $lst = <>$
$f* \bullet \frown/ <>$
$= f * \frown/ <>$ [Functional Composition]
$= f* <>$ [Definition of $\frown/$]
$= <>$ [Definition of $*$]
$= \frown/ <>$ [Definition of $\frown/$]
$= \frown/f ** \ <>$ [Definition of $*$]

Inductive step: We assume that $f* \bullet \frown/ls = \frown/ \bullet f ** ls$.
We must prove that $f* \bullet \frown/(l : ls) = \frown/ \bullet f ** (l : ls)$.

$f* \bullet \frown/(l : ls)$
$= f * \ \frown/(l : ls)$ [Functional Composition]
$= f * (l \frown (\frown/ls))$ [Definition of $\frown/$]
$= f * \ l \frown (f * (\frown/ls))$ [Definition of $*$]
$= f * \ l \frown (f ** ls)$ [Assumption]
$= \frown/(f * \ l : f ** ls)$ [Definition of $\frown/$]
$= \frown/(f ** l : ls)$ [Definition of $*$]
$= \frown/ \bullet f **(l : ls)$ [Functional Composition]

□

D.2 Lemma 8

$$\frown/ \bullet \frown/ = \frown/ \bullet \frown/*$$

This lemma is also proved by induction as seen below:

$\frown/ \bullet \frown/ = \frown/ \bullet \frown/*$

Base Case: lst = <>
$\frown/ \bullet \frown/ <>$
$= \frown/ \bullet \frown/* \ <>$ [Definition of $*$]

Inductive step: We assume that $\frown/ \frown / ls = \frown/ \frown / * \ ls$
We must prove that $\frown/ \bullet \frown/(l : ls) = \frown/ \bullet \frown/ * \ (l : ls)$

$\frown/ \bullet \frown/(l : ls)$
$= \frown/(l \frown (\frown/ls))$ [Definition of $\frown/$]
$= (l \frown (\frown/ \frown /ls))$ [Definition of $\frown/$]
$= (l \frown (\frown/ \frown / * ls))$ [Assumption]
$= \frown/(l : (\frown/ * ls))$ [Definition of $\frown/$]
$= \frown/(\frown/l : (\frown/ * ls))$ [Definition of $\frown/$]
[l is a list of single elements]
$= \frown/ \frown / * (l : ls)$ [Definition of $*$]
$= \frown/ \bullet \frown/ * (l : ls)$ [Functional Composition]

$\square$

D.3 Lemma 9

$$head'\ l = l \quad if\ l =<> \quad or\ \#l = 1$$

head' l=l if $l =<>$ or $\#l = 1$
Case: $l =<>$
$head'\ <>=<>$

Case $\#l = 1$
$l =< e >$ and
$head'\ l =< e >= l$ [Definition of $head'$]

$\square$

D.4 Lemma 10

$$\frown/(l_1 \frown l_2) = \frown/l_1 \frown \frown/l_2$$

This lemma is proved by induction as follows:

$\frown/(l_1 \frown l_2) = \frown/l_1 \frown \frown/l_2$

Base Case:
$l_1 = <>$
$\frown/(<> \frown l_2) = \frown/ <> \frown \frown/l_2$
$= \frown/l_2 = <> \frown \frown/l_2$ [Definition of $\frown$ and $\frown/$]
$= \frown/l_2 = \frown/l_2$ [Definition of $\frown$]

$l_2 = <>$
$\frown/(l_1 \frown <>) = \frown/l_1 \frown \frown/ <>$
$= \frown/l_1 = \frown/l_1 \frown <>$ [Definition of $\frown$ and $\frown/$]
$= \frown/l_1 = \frown/l_1$ [Definition of $\frown$]

Inductive step: We assume that $\frown/(l_1 \frown l_2) = \frown/l_1 \frown \frown/l_2$
We must prove that $\frown/((l : l_1) \frown l_2) = \frown/(l : l_1) \frown \frown/l_2$

$\frown/((l : l_1) \frown l_2)$
$= l \frown \frown/(l_1 \frown l_2)$ [Definition of $\frown/$]
$= l \frown (\frown/l_1 \frown \frown/l_2)$ [Assumption]
$= (l \frown \frown/l_1) \frown \frown/l_2)$ [Property of $\frown$]
$= \frown/(l : l_1) \frown \frown/l_2$ [definition of $\frown/$]

$\square$

D.5 Lemma 11

$$head'(l \frown ((\frown/ \bullet f*)\ l)) = head'\ l$$

This lemma is also proved by induction as seen below:

$head'(l \frown ((\frown/ \bullet f*)\ l)) = head'\ l$

Base Case: $l = <>$
$head'(<> \frown((\frown/ \bullet f*)\ <>))$
$= head'(<> \frown(\frown/ <>)$ [Definition of $*$]
$= head'(<> \frown <>)$ [Definition of $\frown/$]

$= head'(<>)$ [Definition of $\frown$]

Inductive step: We assume that $head'(l \frown ((\frown/ \bullet f*)\ l)) = head'\ l$
We must prove that $head'((x : l) \frown ((\frown/ \bullet f*)\ (x : l))) = head'\ (x : l)$

$head'((x : l) \frown ((\frown/ \bullet f*)\ (x : l)))$
$= < x >$ [Definition of $head'$]
$= head'\ (x : l)$ [Definition of $head'$]

$\square$

D.6 Lemma 12

$$head'(l_1 \frown l_2 \frown l_1) = head'(\frown/ < l_1, l_2 >)$$

This lemma is proved by induction as follows:

$head'(l_1 \frown l_2 \frown l_1) = head'(\frown/ < l_1, l_2 >)$

Base Case:
$l_1 =<>$
$head'(<> \frown l_2 \frown <>) = head'(l_2)$ [Definition of $\frown$]
$= head'(\frown/ << >, l_2 >)$ [Definition of $\frown/$]

$l_2 =<>$
$head'(l_1 \frown <> \frown l_1) = head'(l_1 \frown l_1)$ [Definition of $\frown$]
$= head'(l_1)$ [Definition of $head'$]
$= head'(\frown/ < l_1, <>>)$ [Definition of $\frown/$]

Inductive step: We assume that $head'(l_1 \frown l_2 \frown l_1) = head'(\frown/ < l_1, l_2 >)$
We must prove that $head'((x : l_1) \frown l_2 \frown (x : l_1)) = head'(\frown/ < (x : l_1), l_2 >)$

$head'((x : l_1) \frown l_2 \frown (x : l_1)) = < x >$ [Definition of $head'$]
$= head'((x : l_1) \frown l_2)$ [Definition of $head'$]
$= head'(\frown/ < (x : l_1), l_2 >)$ [Definition of $\frown/$]

$\square$

D.7 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:
$B =$
$< a : as \mid a \leftarrow A, as \leftarrow B > = < a : as \mid a \leftarrow A, as \leftarrow>$
$= <>$ [List Comprehension]
$= << a, b >\mid a \leftarrow A, b \leftarrow>$ [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
$< a : as \mid a \leftarrow A,$
$\quad as \leftarrow \{< x >\} \cup B > \wedge \forall l : \{< x >\} \cup B$
$\quad \bullet \#l = 1 =<< a, b >\mid a \leftarrow A, b \leftarrow \{< x >\} \cup B >$

$< a : as \mid a \leftarrow A, as \leftarrow \{< x >\} \cup B >$
$= < a : as \mid a \leftarrow A, as \leftarrow \{< x >\} >$
$\frown < a : as \mid a \leftarrow A, as \leftarrow B >$ [Property of $\frown$]
$= << a, x >\mid a \leftarrow A > \frown < a : as \mid a \leftarrow A, as \leftarrow B >$ [List Comprehension]
$= << a, b >\mid a \leftarrow A, b \leftarrow \{< x >\} > \frown << a, b >\mid a \leftarrow A, b \leftarrow B >$ [List Comprehension]
[Assumption]
$= << a, b >\mid a \leftarrow A, b \leftarrow \{< x >\} \cup B >$ [Property of $\frown$]

$\square$

D.8 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

$$\begin{array}{l} if\ lname \in \operatorname{dom} \Gamma_L \wedge args \in \operatorname{dom} \Gamma_L\ lname \wedge p' \in \operatorname{dom} \Gamma_L\ lname\ args\ then \\ \quad let\ (newp, proofs) = \Gamma_L\ lname\ args\ p' \ \in \\ \quad < (newp, pr' \cup nproofs) > \\ else \\ \quad <> \end{array}$$

As $pr' \subseteq pr' \cup proofs$ we can conclude that
$\forall (p, pr) \; \in \; [\![\mathbf{law} \;\; l \; (args)]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

$t = \mathbf{skip}$

$[\![\mathbf{skip}]\!] \Gamma_L \; \Gamma_T \; (p', pr') =$ ¡(p',pr')¿
So $\forall (p, pr) \; \in \; [\![\mathbf{skip}]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

$t = \mathbf{fail}$

$[\![\mathbf{fail}]\!] \Gamma_L \; \Gamma_T \; (p', pr') =$ ¡¿
So $\forall (p, pr) \; \in \; [\![\mathbf{fail}]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

$t = \mathbf{abort}$

$[\![\mathbf{abort}]\!] \Gamma_L \; \Gamma_T \; (p', pr') = \perp_{RMCell*}$
So $\forall (p, pr) \; \in \; [\![\mathbf{abort}]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

Inductive step: We assume that
$\forall \, t_1, t_2 : Tactic$ and
$\forall (p, pr) \; \in \; [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$ and
$\forall (p, pr) \; \in \; [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$
We must prove that

$\forall (p, pr) \; \in \; [\![t_1; \; t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

$[\![t_1; \; t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr') = \frown / \bullet ([\![t_2]\!] \Gamma_L \; \Gamma_T) * \bullet [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p', pr')$ [Definition of ;]

let $[\![t_1]\!] \Gamma_L \; \Gamma_T \; (p', pr') = list_A$
By Assumption we have that
$\forall (p'', pr'') \; \in \; list_A \bullet pr' \subseteq pr''$
Now, let $\frown / \bullet ([\![t_2]\!] \Gamma_L \; \Gamma_T) * \;\; list_A = list_B$
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

$\forall (p, pr) \; \in \; [\![t_1 \mid t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

$[\![t_1 \mid t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr')$
$= \frown / < [\![t_1]\!] \Gamma_L \; \Gamma_T, [\![t_1]\!] \Gamma_L \; \Gamma_T >^{\circ} (p', pr')$ [Definition of $|$]
$= ([\![t_1]\!] \Gamma_L \; \Gamma_T \; (p', pr')) \frown ([\![t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr'))$ [Definition of $\circ$ and $\frown/$]
If, by assumption,
$\forall (p, pr) \; \in \; [\![t_1]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$ and
$\forall (p, pr) \; \in \; [\![t_2]\!] \Gamma_L \; \Gamma_T \; (p', pr') \bullet pr' \subseteq pr$

so we can say that
$\forall (p, pr) \ \in \ [\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') \frown [\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

We must also prove that

$\forall (p, pr) \ \in \ [\![!t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$[\![!t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr')$
$= head'([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of !]
$= < head([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') >$ [Definition of $head'$]
If, by assumption, we have that
$\forall (p, pr) \ \in \ [\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$ and
we also can say that
$\forall (p, pr) \ \in < head([\![t_1]\!]\Gamma_L \ \Gamma_T \ (p', pr') > \bullet \ pr' \subseteq pr$ 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

$$\begin{aligned}
&[\![t_1 \boxed{;} t_2]\!]\Gamma_L \ \Gamma_T \ (p_1'; \ p_2', proofs) = \\
&\qquad \Omega_{sc} * (\textstyle\prod(< ([\![t_1]\!]\Gamma_L \ \Gamma_T)(p_1', proofs), \\
&\qquad\qquad ([\![t_2]\!]\Gamma_L \ \Gamma_T)(p_2', proofs) >))
\end{aligned}$$

We know that
$\prod < [\![t_1]\!]\Gamma_L \ \Gamma_T(p_1', proofs), [\![t_2]\!]\Gamma_L \ \Gamma_T(p_2', proofs) >$
$= << (p_1, pr_1), (p_2, pr_2) >$
$\quad | \ (p_1, pr_1) \leftarrow [\![t_1]\!]\Gamma_L \ \Gamma_T(p_1', proofs),$
$\quad (p_2, pr_2) \leftarrow [\![t_1]\!]\Gamma_L \ \Gamma_T(p_2', proofs) >$ [Definition of $\prod$]
$= list_P$
and that
$\Omega_{sc} * \ list_P$
$= < (p_1; \ p_2, pr_1 \frown pr_2)$
$\quad | \ (p_1, pr_1) \leftarrow [\![t_1]\!]\Gamma_L \ \Gamma_T(p_1', proofs),$
$\quad (p_2, pr_2) \leftarrow [\![t_1]\!]\Gamma_L \ \Gamma_T(p_2', proofs) >$ [Definition of Ω_{sc} and $*$]
$= list_\Omega$
By assumption we have that
$\forall (p_1, pr_1) \ \in \ [\![t_1]\!]\Gamma_L \ \Gamma_T \ (p_1', pr') \bullet pr' \subseteq pr_1$ 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 $\forall (p_1; \ p_2, pr_1 \frown pr_2) \ \in \ list_\Omega \bullet pr' \subseteq pr_1 \frown pr_2$

We must also prove that
$\forall (p, pr) \ \in \ [\![\boxed{\textbf{if}}\, tactics \boxed{\textbf{fi}}]\!]\Gamma_L \ \Gamma_T(\textbf{if} \ \ GC \ \ \textbf{fi}, pr') \bullet pr' \subseteq pr$

$= [\![\boxed{\textbf{if}}\, tactics \boxed{\textbf{fi}}]\!] \Gamma_L\, \Gamma_T (\textbf{if}\ \ GC\ \ \textbf{fi}, pr')$
$= \Omega_{if}\ (pr')$
$\quad (mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))$ [Definition of $\boxed{\textbf{if}}$ $\boxed{\textbf{fi}}$]
$= (if\ pr')*$
$\quad (mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))$ [Definition of Ω_{if}]
$= < (\textbf{if}\ \ gc\, \textbf{fi}, pr' \cup nps)$
$\quad | (gc, nps) \leftarrow mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))) >$
[Definition of if]

We can see that
$\forall (p, pr)\ \in < (\textbf{if}\ \ gc\, \textbf{fi}, pr' \cup nps)$
$\quad | (gc, nps) \leftarrow mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))) >$
$\quad \bullet\ pr' \subseteq pr' \cup nps$

We must also prove that
$\forall (p, pr)\ \in\ [\![\boxed{\textbf{do}}\, tactics \boxed{\textbf{od}}]\!] \Gamma_L\, \Gamma_T (\textbf{do}\ \ GC\ \ \textbf{od}, pr') \bullet pr' \subseteq pr$

$= [\![\boxed{\textbf{do}}\, tactics \boxed{\textbf{od}}]\!] \Gamma_L\, \Gamma_T (\textbf{do}\ \ GC\ \ \textbf{od}, pr')$
$= \Omega_{do}\ (pr')$
$\quad (mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))$ [Definition of $\boxed{\textbf{do}}$ $\boxed{\textbf{od}}$]
$= (do\ pr')*$
$\quad (mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))))$ [Definition of Ω_{do}]
$= < (\textbf{do}\ \ gc\, \textbf{od}, pr' \cup nps)$
$\quad | (gc, nps) \leftarrow mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))) >$
[Definition of do]

We can see that
$\forall (p, pr)\ \in < (\textbf{do}\ \ gc\, \textbf{od}, pr' \cup nps)$
$\quad | (gc, nps) \leftarrow mkGC\ (brG\ GC)$
$\quad\quad (\prod (apply\ (([\![\_]\!] \Gamma_L\ \Gamma_T) * tactics)\ (\Phi_r * (brP\ \ GC)))) >$
$\quad \bullet\ pr' \subseteq pr' \cup nps$

We must also prove that
$\forall (p, pr)\ \in\ [\![\boxed{\textbf{var}}\ t_1 \boxed{\ ]\!]\ }]\!] \Gamma_L\, \Gamma_T\ ([\![\textbf{var}\ \ d \bullet p']\!], pr') \bullet pr' \subseteq pr$

$[\![\boxed{\textbf{var}}\ t_1 \boxed{\ ]\!]\ }]\!] \Gamma_L\, \Gamma_T\ ([\![\textbf{var}\ \ d \bullet p']\!], pr')$
$= (var\ \ d) *\ ([\![t_1]\!] \Gamma_L\, \Gamma_T\ (p', pr'))$ [Definition of $\boxed{\textbf{var}}$ $\boxed{\ ]\!]\ }$]
$= < ([\![\textbf{var}\ \ d \bullet p]\!], pr) \mid (p, pr) \leftarrow ([\![t_1]\!] \Gamma_L\, \Gamma_T\ (p', pr')) >$ [Definition of var]

So, we can say that

$\forall (p, pr) \ \in < (\llbracket \textbf{var} \ \ d \bullet p \rrbracket, pr)$

$\qquad\qquad \mid (p, pr) \leftarrow (\llbracket t \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) > \bullet \ pr' \subseteq pr$ [Assumption]

We must also prove that

$\forall (p, pr) \ \in \ \llbracket \boxed{\textbf{cons}} \, t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{cons} \ \ d \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\textbf{cons}} \, t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{cons} \ \ d \bullet p' \rrbracket, pr')$

$= (cons \ \ d) * \ \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\boxed{\textbf{cons}} \boxed{\rrbracket}$]

$= < (\llbracket \textbf{cons} \ \ d \bullet p \rrbracket, pr) \mid (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) >$ [Definition of *cons*]

So, we can say that

$\forall (p, pr) \ \in < (\llbracket \textbf{cons} \ \ d \bullet p \rrbracket, pr)$

$\qquad\qquad \mid (p, pr) \leftarrow (\llbracket t \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) > \bullet \ pr' \subseteq pr$ [Assumption]

We must also prove that

$\forall (p, pr) \ \in \ \llbracket \boxed{\textbf{pmain}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\textbf{pmain}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \bullet p' \rrbracket, pr')$

$= (procm \ pname \ body) * \ \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\boxed{\textbf{pmain}} \ \boxed{\rrbracket}$]

$= < (\llbracket \textbf{proc} \ \ pname = body \bullet p \rrbracket, pr)$

$\qquad \mid (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) >$ [Definition of *procm*]

So, we can say that

$\forall (p, pr) \ \in < (\llbracket \textbf{proc} \ \ pname = body \bullet p \rrbracket, pr)$

$\qquad\qquad \mid (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) > \bullet \ pr' \subseteq pr$ [Assumption]

We must also prove that

$\forall (p, pr) \ \in \ \llbracket \boxed{\textbf{pmainvariant}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \, \textbf{variant} \ \ v \, \textbf{is} \ \ e \bullet p' \rrbracket, pr')$

$\qquad \bullet \ pr' \subseteq pr$

$\llbracket \boxed{\textbf{pmainvariant}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \, \textbf{variant} \ \ v \, \textbf{is} \ \ e \bullet p' \rrbracket, pr')$

$= (variant \ pname \ body \ v \ e) * \ \ (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr'))$ [Definition of $\boxed{\textbf{pmainvariant}} \ \boxed{\rrbracket}$]

$= < (\llbracket \textbf{proc} \ \ pname = body \, \textbf{variant} \ \ v \, \textbf{is} \ \ e \bullet p \rrbracket, pr)$

$\qquad \mid (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) >$ [Definition of *variant*]

So, we can say that

$\forall (p, pr) \ \in < (\llbracket \textbf{proc} \ \ pname = body \, \textbf{variant} \ \ v \, \textbf{is} \ \ e \bullet p \rrbracket, pr)$

$\qquad\qquad \mid (p, pr) \leftarrow (\llbracket t_1 \rrbracket \Gamma_L \ \Gamma_T \ (p', pr')) >$

$\qquad\qquad \bullet \ pr' \subseteq pr$ [Assumption]

We must also prove that

$\forall (p, pr) \ \in \ \llbracket \boxed{\textbf{pbody}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \bullet p' \rrbracket, pr') \bullet pr' \subseteq pr$

$\llbracket \boxed{\textbf{pbody}} \ \ t_1 \boxed{\rrbracket} \rrbracket \Gamma_L \ \Gamma_T \ (\llbracket \textbf{proc} \ \ pname = body \bullet p' \rrbracket, pr')$

$= (procb\ pname\ p') * ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (body, pr'))$ [Definition of $\boxed{\textbf{pbody}}\ \boxed{]\!]}$]
$= < ([\![\textbf{proc}\ pname = body' \bullet p']\!], pr)$
$\quad | (body', pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr')) >$ [Definition of *procb*]

So, we can say that
$\forall (p, pr) \in < ([\![\textbf{proc}\ pname = body' \bullet p']\!], pr)$
$\quad | (body', pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr')) > \bullet\ pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \in [\![\boxed{\textbf{pbodyvariant}}\ t_1 \boxed{]\!]}]\!]\Gamma_L\ \Gamma_T\ ([\![\textbf{proc}\ pname = body\,\textbf{variant}\ v\,\textbf{is}\ e$
$\bullet\ p']\!], pr') \bullet pr' \subseteq pr$

$[\![\boxed{\textbf{pbodyvariant}}\ t_1 \boxed{]\!]}]\!]\Gamma_L\ \Gamma_T\ ([\![\textbf{proc}\ pname = body\,\textbf{variant}\ v\,\textbf{is}\ e \bullet p']\!], pr')$
$= (variantb\ pname\ v\ e\ p') * ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (body, pr'))$ [Definition of $\boxed{\textbf{pbodyvariant}}\ \boxed{]\!]}$]
$= < ([\![\textbf{proc}\ pname = body'\,\textbf{variant}\ v\,\textbf{is}\ e \bullet p']\!], pr)$
$\quad | (body', pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (body, pr')) >$ [Definition of *variantb*]

So, we can say that
$\forall (p, pr) \in < ([\![\textbf{proc}\ pname = body'\,\textbf{variant}\ v\,\textbf{is}\ e \bullet p']\!], pr)$
$\quad | (body', pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (body, pr')) >$
$\quad \bullet\ pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \in [\![\boxed{\textbf{val}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{val}\ v : t \bullet p')(pars), pr') \bullet pr' \subseteq pr$

$[\![\boxed{\textbf{val}}\ t_1]\!]\Gamma_L\ \Gamma_T\ ((\textbf{val}\ v : t \bullet p')(pars), pr')$
$= (val\ v\ t\ pars) * ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr'))$ [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
$\forall (p, pr) \in < ((\textbf{val}\, v : t \bullet p), pr)$
$\quad | (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr')) > \bullet\ pr' \subseteq pr$ [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
$\forall (p, pr) \in < ((\textbf{res}\, v : t \bullet p)(pars), pr)$
$\quad | (p, pr) \leftarrow ([\![t_1]\!]\Gamma_L\ \Gamma_T\ (p', pr')) > \bullet\ pr' \subseteq pr$ [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 $\boxed{\textbf{val\_res}} \ \boxed{[\![\ ]\!]}$]
$= < ((\textbf{val\_res} \, v : t \bullet p)(pars), pr)$
$\quad | \ (p, pr) \leftarrow ([\![t_1]\!] \Gamma_L \ \Gamma_T \ (p', pr')) >$ [Definition of *valres*]

So, we can say that
$\forall (p, pr) \ \in < ((\textbf{val\_res} \, v : t \bullet p)(pars), pr)$
$\quad | \ (p, pr) \leftarrow ([\![t_1]\!] \Gamma_L \ \Gamma_T \ (p', pr')) > \bullet \ pr' \subseteq pr$ [Assumption]

We must also prove that
$\forall (p, pr) \ \in \ [\![\textbf{tactic} \ \ name(\{argument\}^*)]\!] \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$[\![\textbf{tactic} \ \ name(args)]\!] \Gamma_L \ \Gamma_T \ (p', pr') =$
$\quad$ *if* $name \in \operatorname{dom} \Gamma_T \wedge args \in \operatorname{dom} \Gamma_T \ tname$ *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
$= [\![t_1]\!] \Gamma_L \ \Gamma_T \ (p', pr')$ We can say that
$\forall (p, pr) \ \in \ [\![t_1]\!] \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$ [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
$\forall (p, pr) \ \in \ [\![\textbf{applies to} \ \ program \ \ \textbf{do} \ \ t_2]\!] \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$

$[\![\textbf{applies to} \ \ program \ \ \textbf{do} \ \ t_2]\!] \Gamma_L \ \Gamma_T \ (p', pr')$
$= [\![\textbf{con} \ \ program \bullet \ \ equals \ (program, pr') t_2]\!] \Gamma_L \ \Gamma_T \ (p', pr')$ [Definition of **applies to _ do _**]
$= [\![|_{v \in TERM} (equals \ (program, pr'); \ t_2)(v)]\!] \ \Gamma_L \ \Gamma_T]\!] \Gamma_L \ \Gamma_T \ (p', pr')$ [Definition os **con**]
We can say that
$\forall (p, pr) \ \in \ [\![t_1]\!] \Gamma_L \ \Gamma_T \ (p', pr') \bullet pr' \subseteq pr$ [Assumption]

$\square$

D.9 Lemma 15a

$$\prod < lst_a, lst_b > \frown \prod < lst_a, lst_c >= \prod < lst_a, lst_b \frown lst_c >$$

$\prod < lst_a, lst_b > \frown \prod < lst_a, lst_c >$
$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_b >$
$\quad \frown < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_c >$ [Definition of $\prod$]
$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow (e2l * lst_b) \frown (e2l * lst_c) >$ [Property of $\frown$]
$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * (lst_b \frown lst_c) >$ [Property of *]
$= \prod < lst_a, lst_b \frown lst_c >$ [Definition of $\prod$]

$\square$

D.10 Lemma 15b

$$\prod < lst_a, lst_c > \frown \prod < lst_b, lst_c >= \prod < lst_a \frown lst_b, lst_c >$$

$\prod < lst_a, lst_c > \frown \prod < lst_b, lst_c >$
$= < (a : as) \mid a \leftarrow lst_a, as \leftarrow e2l * lst_c >$
$\quad \frown < (a : as) \mid a \leftarrow lst_b, as \leftarrow e2l * lst_c >$ [Definition of $\prod$]
$= < (a : as) \mid a \leftarrow lst_a \frown lst_b, as \leftarrow e2l * lst_c >$ [Property of $\frown$]
$= \prod < lst_a \frown lst_b, lst_c >$ [Definition of $\prod$]

$\square$

D.11 Lemma 16

$$\Omega_{sc} * lst_a \frown \Omega_{sc} * lst_b = \Omega_{sc} * lst_a \frown lst_b$$

$\Omega_{sc} * lst_a \frown \Omega_{sc} * lst_b$
$= < \Omega_{sc}\ a \mid a \leftarrow lst_a > \frown < \Omega_{sc}\ a \mid a \leftarrow lst_b >$ [Definition of *]
$= < \Omega_{sc}\ a \mid a \leftarrow lst_a \frown lst_b >$ [Property of $\frown$]
$= \Omega_{sc} * lst_a \frown lst_b$ [Definition of *]

$\square$

D.12 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 $head'$]

Inductive step: We assume that $head'(f * \ lst) = f * (head' \ lst)$
We must prove that $head'(f * \ (x : lst)) = f * (head'(x : lst))$

$head'(f * \ (x : lst)) = head'(f \ x : f * \ lst)$ [Definition of *]
$= < f \ x >$ [Definition of $head'$]
$= f * \ < x >$ [Definition of *]
$= f * (head'(x : lst))$ [Definition of $head'$]

$\square$

D.13 Lemma 18

$$head'(\prod(llst)) = \prod(head' * \ llst)$$

This lemma is also proved by induction as follows:

Base Case:
$llst = <>$
$head'(\prod(llst))$
$= head'(\prod(<>))$
$= head'(<>)$ [Definition of $\prod$]
$= <>$ [Definition of $head'$]
$= \prod(<>)$ [Definition of $\prod$]
$= \prod(head' * \ <>)$ [Definition of $*$]

Inductive step: We assume that $head'(\prod(ls)) = \prod(head' * \ ls)$
We must prove that $head'(\prod(l : ls)) = \prod(head' * \ (l : ls))$

$head'(\prod(l : ls)) = head'(< a : as \mid a \leftarrow l, as \leftarrow \prod ls >)$ [Definition of $\prod$]
$= < a : as \mid a \leftarrow head' \ l, as \leftarrow \prod \ head' * \ ls >)$ [Property of List Comprehension]
$= \prod(head' \ l : head' * \ ls)$ [Definition of $\prod$]
$= \prod(head' * \ l : ls)$ [Definition of $*$]

$\square$

D.14 Lemma 19

$$head'(head'(lst)) = head'(lst)$$

This lemma is proved by induction as seen below:

Base Case:
$lst = <>$
$head'(head' \ <>)$
$= head'(<>)$ [Definition of $head'$]

Inductive step: We assume that $head'(head'(xs)) = head'(xs)$
We must prove that $head'(head'(x : xs)) = head'(x : xs)$

$head'(head'(x : xs))$
$= head' \ < x >$ [Definition of $head'$]
$= head' \ x : xs$ [Definition of $head'$]

□

D.15 Lemma 20

$$\frown / \bullet t * \ lst \neq <> \Rightarrow \frown / \bullet (succs \ t) * \ lst \neq <>$$

$\frown / \bullet t * \ lst \neq <>$
$\Rightarrow \exists r : RMCell \bullet \exists i : \mathbb{N} \bullet lst[i] = r \wedge t \ r \neq <>$
$\Rightarrow succs \ t \ r \neq <>$
$\Rightarrow \frown / \bullet (succs \ t) * \ lst \neq <>$

It must exist an r in lst such that $t \ r \neq <>$

Base Case:
$lst = <>$
$head'(head' \ <>)$
$= head'(<>)$ [Definition of $head'$]

Inductive step: We assume that $head'(head'(xs)) = head'(xs)$
We must prove that $head'(head'(x : xs)) = head'(x : xs)$

$head'(head'(x : xs))$
$= head' \ < x >$ [Definition of $head'$]
$= head' \ x : xs$ [Definition of $head'$]

□

References

[1] R. J. R. Back. Procedural Abstraction in the Refinement Calculus. Technical report, Department of Computer Science, Åbo - Finland, 1987. Ser. A No. 55.

[2] A. L. C. Cavalcanti, A. Sampaio, and J. C. P. Woodcock. Procedures, Parameters, and Substitution in the Refinement Calculus. Technical Report TR-5-97, Oxford University Computing Laboratory, Oxford - UK, February 1997.

[3] 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, 1999.

[4] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1nd edition, 1990.

[5] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[6] J. Goguen, A. Stevenes, K. Hobley, and H. Hilberdink. 2OBJ, A Metalogical Framework Based on Equational Logic. *Philosophical Transactions of the Royal Society, Series A*, 339:69 – 86, 1992.

[7] 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.

[8] A. Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice-Hall, 1990.

[9] A. P. Martin, P. H. B. Gardiner, and J. C. P. Woodcock. A Tactical Calculus. *Formal Aspects of Computing*, 8(4):479–489, 1996.

[10] C. C. Morgan. *Programming from Specifications*. Prentice-Hall, 2nd edition, 1994.

[11] L. C. Paulson. *ML for the Working Programmer*. CUP, 1991.