
ELSEVIER

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 93 (2004) 118–137
www.elsevier.com/locate/entcs

A Calculus of Tactics and Its Operational Semantics

G.I. Jojgov [1]

Technische Universiteit Eindhoven
The Netherlands

H. Geuvers[2]

Katholieke Universiteit Nijmegen
The Netherlands

Abstract

This paper presents work in progress on a calculus of tactics for a hypothetical interactive theorem prover based on a λ-calculus for higher-order logic (λHOL). The calculus of tactics is an extension of a calculus of open terms for λHOL. In contrast to other systems where the semantics of tactics is given by the semantics of their implementation in a general programming language (e.g. OCAML) we are able to define what a tactic does in terms of the state of the theorem prover expressed by an open term that encodes the incomplete proof created so far at that given state.
We present typed operational semantics for the tactics calculus and show that it is sound and complete with respect to the calculus of open terms. The soundness theorem goes further to establish the relation between the states of the prover before and after the execution of a tactic.

Keywords: interactive theorem proving. type theory, tactics, operational semantics

1 Introduction and Motivation

Proof assistants are computer programs that facilitate the formalization and the development of complex proofs and the theory required for them. In proof development, as in computer programming, we can distinguish between declarative and procedural approaches. In the declarative approach to proof

[1] Email: G.I.Jojgov@tue.nl
[2] Email: herman@cs.kun.nl

doi:10.1016/j.entcs.2003.12.031

construction one records consecutive statements for which the assistant can check that follow from the previous ones. The procedural approach is taken by the tactics-based proof assistants where one issues commands that manipulate proofs until a proof of the wanted goal is constructed. These commands, often referred to as *tactics*, can be seen as procedures that find instantiations of open goals in the current state of the prover. The complexity of the tactics may vary from applications of a single derivation rule to complicated (semi)decision procedures for (semi)decidable classes of problems.

Often the assistant provides a meta-language that allows the user to define more complex tactics from the existing ones by means composition, iteration, etc. The operators of this language are called *tacticals*.

These tactics and tacticals are the building blocks for the management of mathematical content on a microscopic level in a proof assistant. Other tools may be available for dealing with documents and files, but to really "navigate" through and extend the body of (formal) mathematics, one uses tactics (and tacticals). It is important to get a more abstract – mathematical as opposed to proof-assistant-oriented – view on tactics and to understand well what the basic microscopic steps are and how they can be composed. In that respect we feel that the current work is contributing to the MKM community.

This paper presents a calculus based on a λ-calculus with open terms for higher-order logic whose terms will be interpreted as tactics. The calculus of open terms and the calculus of tactics form the first two layers in a model of a hypothetical tactics-based interactive theorem prover for higher-order logic. The third layer of the model – the layer of the tacticals – is also a subject of interest but it falls outside the scope of this paper. After presenting this model and describing the language of tactics we define a typed operational semantics for the tactics calculus and show that it is sound and complete with respect to the calculus of open terms.

Why do we need a calculus of tactics and what are the benefits from defining a semantics for it? First, it provides a dedicated languages that is independent of the implementation language of the prover. This allows users to write portable tactics that do not need to be recompiled with each new version. From a theoretical point of view the interest is even bigger because the semantics of the tactics calculus provides a direct connection between the definition of the tactic and the manipulation of the proof states of the prover that it describes. This provides increased reliability in comparison with the case when tactics are written in the implementation language of the prover. Furthermore, a semantics of the tactics language would allow us to reason about tactics. For example, after defining a tactic, one could prove that it always terminates or that it solves certain classes of goals.

The introduction of the tactic calculus and the definition of its semantics go further than syntactic packaging of the underlying system of open terms in the sense that they allow us to define proof-search procedures as terms in the tactic calculus. In these search procedures we may use constructs not available in the underlying calculus as unification, failure handling and recursion. The idea is that the result of such procedures should be related to the calculus of open terms in order to guarantee that the process of interactive proof construction is sound. Next to stating that successful evaluations of tactics produce a well-typed result, the soundness theorem gives a relation between the state of the prover before and after the execution of a tactic. This means for example that when we execute a tactic that is supposed to solve a goal, this does not happen by just introducing a new axiom.

With respect to earlier work on tactic languages like the language $\mathcal{L}_{Tac}$ [2] of tactics for the theorem prover Coq, this paper can be seen as complimentary. In $\mathcal{L}_{Tac}$ the basic tactics provided by Coq are taken as primitives from which new tactics can be built, while we aim at explaining these primitives by codifying the manipulations of proof states. Of course the relation with Coq is only conceptual, we do not claim to give semantics to the primitive tactics of Coq because Coq is based on the Calculus of Inductive Constructions (CIC) while we work with λHOL and we do not even try to handle inductive types.

The paper is organized as follows: Section 2 introduces the model of the theorem prover that we work with by describing its three layers. When discussing the layer of tactics in Section 2.3 we introduce the syntax of the tactic calculus. Then in Section 3 we show how some common tactics are definable in it. Section 4 introduces the semantics of the tactic calculus and shows its adequacy. We conclude with notes on related and future work in Section 5.

2 Modelling an Interactive Prover

Looking at interactive proof assistants that take the procedural approach to proof construction (like Coq [10] and Lego [8] for example) we can clearly identify three levels of abstraction – the object layer that represents the proof objects, the tactic layer of the commands that do proof search or apply deduction rules and the top layer of tacticals that acts as an interface to the user and allows us to define new tactics by composing existing ones (see Figure 1). Below we will briefly comment on each of the layers in order to introduce them formally and fix the terminology.

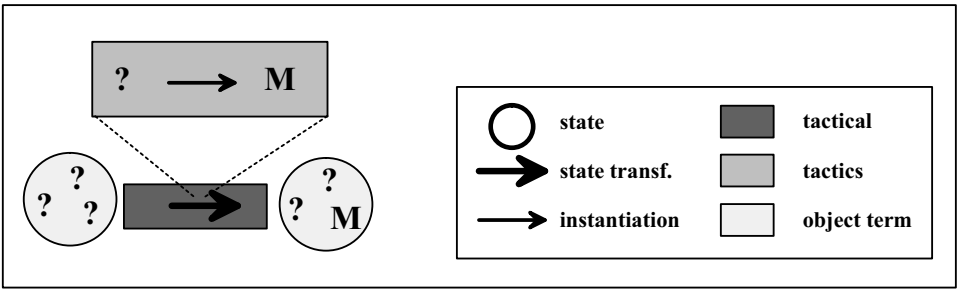

Fig. 1. States, represented in the object layer, are transformed by tacticals. The tacticals use tactics to find instantiations of goals in the states.

2.1 Object Layer

The object layer in our model is a λ-calculus for higher-order logic (λHOL) extended with open terms. The choice of λHOL is not really relevant, we could have built the model with another λ-calculus that admits extensions with open terms. Open terms are terms containing placeholders for unknown terms that we are trying to find. An open term typically stands for an incomplete proof that is being developed in the prover. We take the approach of [3] to open terms where the extension of λHOL with open terms is done by adding special parameterized variables (*meta-variables*), rules for typing them and an instantiating operation that allows us to fill in a value for an unknown. For details on the calculus of open terms we refer the reader to [3] [3]. Here we will only note that a meta-variable has a declaration $?m[x_1 : A_1 \dots x_n : A_n] : A$ (typically given in a context) and instances $m[t_1 \dots t_n]$ that occur in terms.

The reader may ask the question why do we need to introduce meta-variables and not just use free variables to represent unknowns. If the objects that a logical formula talks about are the object level, then the logical reasoning is done on the meta-level. When doing interactive proof construction in a calculus of open terms, the proofs are our objects and we actually work on the meta-meta-level. There are serious reasons to distinguish between dependencies that originated from the object level and those coming from the meta-level. For example the goal to prove $A \to A$ and the goal of proving A under assumption A may be logically equivalent (and they should be, if we want soundness of the proof construction), but from the meta-meta-viewpoint they are different, because one way to prove $A \to A$ could be to get it by the introduction rule for $\to$ and by performing this proof construction step we arrive at the goal of proving A under assumption A. But we could have

[3] For convenience of referees, the typing rules for the calculus of open terms are included in Appendix A

as well applied another logical rule and constructed a different proof that was not obtainable had we started from the second goal. By introducing meta-variables we can make a distinction between object-level dependencies (i.e. a functional variable $f : \sigma \to \sigma$) and meta-level dependencies ($m[x:\sigma] : \sigma$). This distinction is necessary in our case because tactic terms define manipulations on open terms, i.e. the tactic calculus is even one more meta-level up.

Apart from the need to distinguish between object- and meta-level dependencies, using meta-variables has some practical advantages. For example, in a first order setting, the introduction of higher-order functions may be undesirable; the consistency of some typing systems is sensitive to the extension of their function space, etc.

2.2 States. Transformations between States.

The state of the prover at a given moment can be represented by a typing judgment in the calculus of open terms. A judgment $\Gamma \vdash M : A$ asserts that the incomplete term M is of type A in context Γ. The term M is the incomplete proof we are developing and Γ contains all declarations of the language that we work in, the global assumptions and the meta-variables representing the unknown parts of the proof. The fact that the judgment is derivable in the calculus of open terms guarantees the 'soundness' of the state in the sense that if we find instantiations of the meta-variables in Γ and propagate them over M we will get a typable term of the type we want to inhabit.

Generally the proof-construction process will be initiated by starting with a judgment of the form $\Gamma_0, ?m_0[\,] : A \vdash m_0[\,] : A$ where Γ_0 is the context in which we want to find an inhabitant of type A. By applying a tactic to the goal m_0 we find an instantiation M_0 for it that possibly contains fresh meta-variables $m_1[\Delta_1] : A_1 \ldots m_n[\Delta_n] : A_n$. The judgment that represents this state of the prover after the tactic application is then

$$\Gamma_0, ?m_1[\Delta_1] : A_1 \ldots ?m_n[\Delta_n] : A_n \vdash M_0 : A$$

The state transformation given by the tactic can be seen in the following way: starting with the original judgment $\Gamma_0, ?m_0[\,] : A \vdash m_0[\,] : A$ we weaken it with the declarations of the new goals: $\Gamma_0, ?m_1[\Delta_1] : A_1 \ldots ?m_n[\Delta_n] : A_n, ?m_0[\,] : A \vdash m_0[\,] : A$. Then we check that the instantiating term M_0 produced by the tactic is of the right type in the context of the new goals, i.e. we make sure that $\Gamma_0, ?m_1[\Delta_1] : A_1 \ldots ?m_n[\Delta_n] : A_n \vdash M_0 : A$. If this is the case, we can use the Cut Lemma[4] of the calculus of open terms to instantiate m_0 and obtain

[4] see Appendix A.

the final judgment.

Therefore we can see the state transformation steps induced by tactic applications as mappings between judgments obtained by composing meta-variable weakening and cutting. The basic properties of the the open terms calculus guarantee that such transformations map derivable judgments into derivable judgments. So if we start with a judgment of the form $\Gamma_0, ?m_0[\,] : A \vdash m_0[\,] : A$ and by transformation steps we manage to reach a judgment with no meta-variables in the context, then we have found an inhabitant of the type A in the context Γ_0.

2.3 Tactics Layer

The tactics layer is a calculus that allows us to define sound transformation steps. To be more precise, in our setting a tactic is a term that when evaluated in the context of a goal produces an instantiation for that goal and possibly new goals used in this instantiation.

Please note that a tactic term only computes the instantiating term and the new goals. It does not change the goal it is intended to solve. The *application* of a tactic to a goal is the operation that actually transforms the proof state. This meta-operation is an example of a *tactical* (see the layer of tacticals below).

The tactics calculus is an extension of the calculus of the object layer and adds to it the following proof-search constructs:

- Introduction of a new goal. This is done by means of the ?-binder. The term $?m\!:\!A.T$ intuitively means "introduce a new meta-variable m of type A in the current context and proceed with T" (m may occur in T). Using a binder is a clean way of introducing fresh names in the presence of the usual bound variable convention.
- Explicit unification constraint. The unification constraints have the following shape: $[M \sim_\Sigma N].T$ where M and N are open terms, Σ is a list of meta-variables and T is a tactic term. The idea of this construct is that we try to unify M and N and if this is successful we proceed with T. The unification, if successful, will provide an instantiation of some meta-variables. This instantiation is applied to T before evaluating it. Σ specifies meta-variables that need to be introduced locally exclusively for the purposes of the unification constraint. For example, if we want to check if φ is an implication, we need to introduce fresh meta-variables A and B and find instantiations for them such that φ unifies with $A \to B$

$$[\varphi \sim_{A,B:\mathsf{Prop}} A \to B].T$$

'Matching variables' like A and B that necessarily need to be instantiated in order to satisfy the constraint are given in Σ.

- Failing tactic. The special tactic Fail represents the tactic that always fails.
- Case distinction. The operator for case distinction allows us to to handle failing tactics by specifying alternative action in case a tactic fails. To evaluate(M else N) we first evaluate M. If it is successful then its value becomes the value of (M else N). Otherwise the value of N is taken.
- Non-terminating recursion. We will allow tactics to be defined by recursion without imposing syntactic constraints that guarantee termination. There are two reasons for allowing non-restricted form of recursion. The first one reason is that a terminating system will necessarily be incomplete and the second is that in the presence of the unification constraints the formulation of a syntactic criterium that guarantees termination seems to be a difficult problem to which we have not found a satisfactory solution (i.e. a decidable syntactic criterium that covers the examples that we intend to give semantics to).

The syntax of the calculi of the object and tactic layers is summarized on Figure 2. For technical reasons we allow also the introduction of global definitions in the contexts (for example $\Gamma, !m[\Delta] := N : A, \Gamma'$) because they provide a convenient way of instantiating meta-variables. When we want to instantiate $m[\Delta]$ by N we just change the declaration $?m[\Delta] : A$ to a definition $!m[\Delta] := N : A$. This saves us the effort of propagating the instantiation through the rest of the context and allows cleaner formulation of the properties of the semantics.

sorts	$\mathcal{S} ::= \mathsf{Prop} \mid \mathsf{Type} \mid \mathsf{Kind}$
object term	$\mathcal{B} ::= x \mid m[\mathcal{B} \ldots \mathcal{B}] \mid \mathcal{B}\mathcal{B} \mid \lambda x{:}\mathcal{B}.\mathcal{B} \mid \Pi x{:}\mathcal{B}.\mathcal{B} \mid \mathcal{S}$
parameter list	$\Delta ::= \varepsilon \mid \Delta, x{:}\mathcal{B}$
meta-variable list	$\Sigma ::= \varepsilon \mid \Sigma, ?m[\Delta]{:}\mathcal{B}$
tactic term	$\mathcal{T} ::= x \mid m[\mathcal{B} \ldots \mathcal{B}] \mid t[\mathcal{B} \ldots \mathcal{B}] \mid \mathcal{T}\mathcal{T} \mid \lambda x{:}\mathcal{B}.\mathcal{T} \mid \Pi x{:}\mathcal{B}.\mathcal{T}$
	$\mid ?m[\Delta]{:}\mathcal{B}.\mathcal{T} \mid [\mathcal{B} \sim_\Sigma \mathcal{B}].\mathcal{T} \mid (\mathcal{T}, \mathcal{T}) \mid \mathsf{Fail}$
defined tactic	$t ::= t[x{:}\mathcal{B} \ldots x{:}\mathcal{B}] := \mathcal{T}$
context	$\mathcal{C} ::= \varepsilon \mid \mathcal{C}, x{:}\mathcal{B} \mid \mathcal{C}, ?m[\Delta]{:}\mathcal{B} \mid \mathcal{C}, !m[\Delta] := \mathcal{B}{:}\mathcal{B}$

Fig. 2. Syntax

2.4 *Example: The tactic* Apply.

The tactic Apply tries to solve a goal φ by specializing a theorem ψ with proof M. For example, if φ is $P(t)$ and ψ is $\forall x.P(x)$ then the tactic produces the instantiation $(M\,t)$ which is a proof of $P(t)$.

In more complicated cases it may introduce new meta-variables for unknown terms needed to instantiate ψ. For example, let a, b and c be terms of type U and R be a binary relation on U. Let M be the proof that R is transitive. Then an application of Apply with argument M to the goal $R(a,c)$ would produce new meta-variables $y[\,]:U$, $p[\,]:R(a,y)$ and $q[\,]:R(y,c)$ and instantiation term $(M\,a\,y[\,]\,c\,p[\,]\,q[\,])$ that can be used to solve the original goal.

To further illustrate the complex behaviour required from Apply, let us assume that there is also a term N representing the proof of $\forall x.R(x,b)$. If we use Apply with N on the goal p we observe two things – first, we see that Apply needs unification (as opposed to matching) to solve the goal because it needs to unify $R(a,y)$ and $R(x,b)$ where both x and y are unknowns. Second, after the instantiation found by unification affects the goal y. Therefore *as a side effect a tactic may force other goals to be solved.*

Using the tactics calculus introduced above we can define of Apply as follows:

$$
\begin{array}{l}
\mathsf{Apply}[\varphi:\mathsf{Prop},\psi:\mathsf{Prop},M:\psi] := \\
\quad [\varphi\sim\psi].M \\
\quad \mathsf{else} \\
\quad\quad [\psi\sim_{?A,?B:\mathsf{Prop}}\Pi x:A.B].?m:A.\mathsf{Apply}[\psi,B,(Mm)] \\
\quad \mathsf{else} \\
\quad\quad [\psi\sim_{?U:\mathsf{Type},?B[x:U]:\mathsf{Prop}}\Pi x:U.B[x]].?m:A.\mathsf{Apply}[\psi,B[m],(Mm)]
\end{array}
$$

We have dropped the brackets around else by defining it as left-associative. On the second line we test whether the goal φ and the theorem ψ can be unified. If yes, we can get a proof of φ from M by applying to it the unifier found by the unification. If the unification fails, we are on the fourth line where we test whether ψ an implication $A\to B$. If this is the case, we introduce a fresh proof meta-variable m of type A and use it to construct the proof $(M\,m)$ of B required to make the recursive call. In the last case we check whether ψ is a universally quantified formula. If it is, we again introduce a fresh meta-variable that we use to eliminate the universal quantifier in the recursive call. Please note that in this case the meta-variable may appear in the type $B[m]$.

Coming back to the transitive relation R, we would expect that when

evaluating the term

$$\mathsf{Apply}[R(a,c), \Pi x,y,z{:}U.R(x,y) \to R(y,z) \to R(x,z), M]$$

we would have to do roughly the following steps (see Section 4 for precise formulation):

$$\begin{aligned}
?x{:}U &\ \mathsf{Apply}[R(a,c), \Pi y,z{:}U.R(x,y) \to R(y,z) \to R(x,z), (M\,x)]\\
?x,y{:}U &\ \mathsf{Apply}[R(a,c), \Pi z{:}U.R(x,y) \to R(y,z) \to R(x,z), (M\,x\,y)]\\
?x,y,z{:}U &\ \mathsf{Apply}[R(a,c), R(x,y) \to R(y,z) \to R(x,z), (M\,x\,y\,z)]\\
?x,y,z{:}U.?p{:}R(x,y) &\ \mathsf{Apply}[R(a,c), R(y,z) \to R(x,z), (M\,x\,y\,z\,p)]\\
?x,y,z{:}U.?p{:}R(x,y).?q{:}R(y,z) &\ \mathsf{Apply}[R(a,c), R(x,z), (M\,x\,y\,z\,p\,q)]\\
?x,y,z{:}U.?p{:}R(x,y).?q{:}R(y,z) &\ [R(a,c) \sim R(x,z)](M\,x\,y\,z\,p\,q)\\
?y{:}U.?p{:}R(a,y).?q{:}R(y,c) &\ (M\,a\,y\,c\,p\,q)
\end{aligned}$$

At the last step we see another potential problem – the constraint $[R(a,c) \sim R(x,z)](M\,x\,y\,z\,p\,q)$ produces an instantiation that affects not only $(M\,x\,y\,z\,p\,q)$ but the whole state.

In general, the evaluation of one subterm of a tactic term may affect whether and to what another subterm evaluates. For example, in the context $A{:}\mathsf{Prop}, g{:}A \to A, ?m[\,]{:}\mathsf{Prop}$ the term $([A \sim A].\lambda f{:}A \to A.f)(\lambda x{:}m[\,].gx)$ cannot be evaluated to a well-typed term while the term $([m[\,] \sim A].\lambda f{:}A \to A.f)(\lambda x{:}m[\,].gx)$ evaluates to $(\lambda f{:}A \to A.f)(\lambda x{:}A.gx)$ (after expansion of definitions).

Hence we should make sure that the semantics correctly captures the side-effects of the evaluation of tactic terms.

2.5 The layer of tacticals

A tactical can generally be described as a mapping that transforms proof states by solving and introducing meta-variables generated by tactic terms. The most basic tactical is the tactic application tactical. It takes a tactic term as an argument, evaluates it in the context of the current goal and instantiates and introduces meta-variables as prescribed by the tactic. Other tacticals include different kinds of composition, failure handling, tacticals for handling of naming etc.

Unfortunately, due to space restrictions we cannot go into a detailed discussion on tacticals and their semantics. In Section 4.1 we will give an impression of the intended semantics of the tactic application tactical because it is essential for our discussion on the semantics of tactic terms.

3 Defining Some Basic Tactics

Below we give (recursive) definitions in our language that represent some common tactics in tactic-based interactive theorem provers. As a matter of convention, the first parameter of the tactics will always be the goal that the tactic solves.

$$\mathsf{Cut}[\varphi:\mathsf{Prop},\psi:\mathsf{Prop}] := ?p:\psi\to\varphi.?q:\psi.(p\,q)$$

$$\mathsf{Assert}[\varphi:\mathsf{Prop},\psi:\mathsf{Prop}] := ?q:\psi.?p:\psi\to\varphi.(p\,q)$$

The tactics Cut and Assert solve the goal φ by introducing a statement ψ and proving $\psi\to\varphi$ and ψ. The difference between the two is the order in which the new goals are generated.

$$\begin{array}{l}\mathsf{Intros}[\varphi:\mathsf{Prop}] := \\ \quad [\varphi \sim_{A,B:\mathsf{Prop}} \Pi x:A.B].\lambda x:A.\mathsf{Intros}[B] \\ \quad else \\ \quad [\varphi \sim_{A:\mathsf{Type},B[xA]:\mathsf{Prop}} \Pi x:A.B[x]].\lambda x:A.\mathsf{Intros}[B[x]] \\ \quad else \\ \quad ?m:\varphi.m\end{array}$$

The tactic Intros takes a goal and as long as possible tries to apply introduction rules to it. For example, $\mathsf{Intros}[\forall x^U.P(x)\to Q(x)]$ results in the term $\lambda x:U.\lambda p:P(x).m[x,p]$ where m is a new meta-variable with declaration

$$m[x:U,p:P(x)]:Q(x)$$

$$\begin{array}{l}\mathsf{Apply}[\varphi:\mathsf{Prop},\psi:\mathsf{Prop},M:\psi] := \\ \quad [\varphi\sim\psi].M \\ \quad else \\ \quad [\psi \sim_{?A,?B:\mathsf{Prop}} \Pi x:A.B].?m:A.\mathsf{Apply}[\psi,B,(Mm)] \\ \quad else \\ \quad [\psi \sim_{?A:\mathsf{Type},?B[xA]:\mathsf{Prop}} \Pi x:A.B[x]].?m:A.\mathsf{Apply}[\psi,B[m],(Mm)]\end{array}$$

We have already seen the tactic Apply in Section 2.4. It tries to produce a term of type φ by specializing the theorem ψ with proof M.

$$\begin{array}{l}\mathsf{Generalize}[\varphi : \mathsf{Prop}, U : \mathsf{Type}, t{:}U] := \\ \quad [B[t] \sim_{?B[x{:}U]\mathsf{Prop}\ \varphi}].?m{:}\Pi x{:}U.B[x].mt\end{array}$$

The tactic Generalize is in a sense the opposite of Apply. Given the goal φ and a term t it produces a new goal $\forall x.B[x]$ such that φ is its specialization by t (i.e. such that $B[t] \equiv \varphi$). This tactic is used in cases it is easier to prove a statement for all values of a variable rather than for a specific one. Its effectiveness is heavily influenced by the power of the unification mechanism one uses.

4 Operational Semantics

So far we have introduced the abstract model of the interactive theorem prover and the syntax of the language of the tactics. In this section we will present rules that allow us to 'execute' tactic terms, or in other words to compute the result of applying a tactic to a state.

As already mentioned, a successful tactic evaluation should produce a term that is intended to instantiate a given goal. The evaluation may have side effects in the form of the introduction of new goals and/or instantiation of pre-existing and even of newly introduced goals.

Given a state $\Gamma, ?m[\Delta]{:}A, \Gamma' \vdash M : B$, the context Γ represents the state-context of the goal m. The context Δ will be called local context of m. When we apply a tactic term t to the goal, we will get a term N that should be of type A in the context of the goal. What complicates the problem are the non-local effects that unification constraints may have. This means that the context Γ of the goal may change after we evaluate the tactic term because some of its meta-variables have been solved or new ones are introduced. Therefore we will use judgments like:

$$\models \langle \Gamma; \Delta \rangle \rhd t \Rightarrow \langle \Gamma'; \Delta' \rangle \rhd N : A$$

This judgment should be read as follows: "When in a state-context Γ and local context Δ we evaluate the tactic term t we obtain the term N of type A in a new state-context Γ' and a new local context Δ'."

As suggested by its name, Δ contains only variable declarations. The meta-variables are declared in the state-context. As an example, consider the following judgment:

$$\models \langle A{:}\mathsf{Prop}; \varepsilon \rangle \rhd \mathsf{Intros}[A \to A] \Rightarrow \langle A{:}\mathsf{Prop}, ?n[x{:}A]{:}A; \varepsilon \rangle \rhd \lambda x{:}A.n[x] : A \to A$$

In the state context $A : \mathsf{Prop}$ and empty local context we evaluate the tactic $\mathsf{Intros}[A \to A]$. As a result we get a new state context with a fresh meta-variable $n[x:A] : A$ and the term $\lambda x:A.n[x]$ of type $A \to A$.

4.1 Tactic application

Before defining the semantic evaluation relation $\models$ we will show how we intend to use it in the tactics application tactical that actually applies a tactic to a proof state.

Let $\Gamma_1, ?m[\Delta] : A, \Gamma_2 \vdash M : B$ be the current state. Suppose we want to apply the tactic term t to m. First we evaluate t in state context Γ_1 and local context Δ. The definition of $\models$ is such that the local context is preserved:

$$\models \langle \Gamma_1; \Delta \rangle \rhd t \Rightarrow \langle \Gamma_1'; \Delta \rangle \rhd N : C$$

Next, we check whether in context Γ_1', Δ the expected type A is $\beta\delta$-equal to the actual type C. If this is the case, then we construct the new state:

$$\Gamma_1', m[\Delta] := N : A, \Gamma_2 \vdash M : B$$

Note that the semantics must ensure that in context Γ_1', Δ the term N is really of type C and that Γ_1' contains all meta-variables of Γ_1 (some of the meta-variables may be converted to definitions and new ones may be added) and exactly the variables of Γ_1. Together, these requirements form the soundness criteria for our semantics (see Theorem 4.8).

4.2 Operational Semantics of Tactics Terms

There are two kinds of judgments. The judgment $\models \langle \Gamma; \Delta \rangle \rhd t \Rightarrow \langle \Gamma'; \Delta \rangle \rhd \mathsf{Fail}$ represents failed evaluation and the judgment $\models \langle \Gamma; \Delta \rangle \rhd t \Rightarrow \langle \Gamma'; \Delta \rangle \rhd N : A$ represents successful evaluation of t to the term N of type A in the new state context Γ' and local context Δ.

The semantics is parameterized by a unification oracle. This oracle is an external function $uni(\Gamma, \Sigma, M, N)$ that will be used to solve typed unification problems. We assume that $uni()$ is a total recursive function that either produces a unifier for M and N in context Γ, Σ or the value Fail indicating that no unifier was found. We note that failure to find a unifier does not necessarily mean that it does not exist because of the undecidability of higher-order unification. For the reasons given in Section 2.3 we separate the meta-variable context Σ representing the 'matching variables' and require that after applying the unifier to M and N no meta-variable of Σ may occur in them.

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd M \Rightarrow \langle \Gamma';\Delta\rangle \rhd P : A}{\models \langle \Gamma;\Delta\rangle \rhd (M \ \mathsf{else}\ N) \Rightarrow \langle \Gamma';\Delta\rangle \rhd P : A} \quad (\mathsf{else}^+)$$

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd M \Rightarrow \langle \Gamma';\Delta\rangle \rhd \mathsf{Fail} \qquad \models \langle \Gamma;\Delta\rangle \rhd N \Rightarrow \langle \Gamma'';\Delta\rangle \rhd Q : A}{\models \langle \Gamma;\Delta\rangle \rhd (M \ \mathsf{else}\ N) \Rightarrow \langle \Gamma'';\Delta\rangle \rhd Q : A} \quad (\mathsf{else}^-)$$

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd \mathsf{Prop} \Rightarrow \langle \Gamma;\Delta\rangle \rhd \mathsf{Prop} : \mathsf{Type} \qquad uni((\Gamma,\Delta),\Sigma,M,N) = \sigma \qquad \models \langle \sigma(\Gamma);\Delta\rangle \rhd \sigma_{|\Sigma}(P) \Rightarrow \langle \Gamma';\Delta\rangle \rhd Q : A}{\models \langle \Gamma;\Delta\rangle \rhd [M \sim_\Sigma N].P \Rightarrow \langle \Gamma';\Delta\rangle \rhd Q : A} \quad (uni^+)$$

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd \mathsf{Prop} \Rightarrow \langle \Gamma;\Delta\rangle \rhd \mathsf{Prop} : \mathsf{Type} \qquad uni((\Gamma,\Delta),\Sigma,M,N) = \mathsf{Fail}}{\models \langle \Gamma;\Delta\rangle \rhd [M \sim_\Sigma N].P \Rightarrow \langle \Gamma;\Delta\rangle \rhd \mathsf{Fail}} \quad (uni^-)$$

$$\frac{\models \langle \Gamma, ?m_1[\Delta]:A;\Delta\rangle \rhd M[m_1[\Delta]/m] \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : B}{\models \langle \Gamma;\Delta\rangle \rhd ?m:A.M \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : B} \quad (MV_{intro}) \quad m_1 \text{ is fresh}$$

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd t_i \Rightarrow \langle \Gamma;\Delta\rangle \rhd t_i : A_i[t_j/x_j]_{j<i} \qquad \models \langle \Gamma;\Delta\rangle \rhd M[\vec{t}/\vec{x}] \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : B}{\models \langle \Gamma;\Delta\rangle \rhd T[t_1 \ldots t_n] \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : B} \quad T[\vec{x}:\vec{A}] := M \text{ defined tactic}$$

$$\models \langle \Gamma;\Delta\rangle \rhd \mathsf{Fail} \Rightarrow \langle \Gamma;\Delta\rangle \rhd \mathsf{Fail} \quad (\mathsf{Fail})$$

$+$ Fail *compatibility rules*

Fig. 3. Rules for the tactic-level calculus.

The definition of $\models$ for the tactic-specific term constructors is given on Figure 3. The rules for the rest of the term constructors are presented in Appendix B. We will briefly comment on the rules of Figure 3.

The rule (else^+) treats the case when the first component of (M else N) successfully evaluates to a term. Then that term is also the value of (M else N).

The case when M fails is covered by the rule (else^-). Then the result of the evaluation of N becomes the value of $(M \ \mathsf{else} \ N)$.

The rules (uni^+) and (uni^-) treat the evaluation of unification constraints. The first premise in both rules is there just to ensure that $\Gamma; \Delta$ are well-typed contexts and play no futher role. The unification function $uni((\Gamma, \Delta), \Sigma, M, N)$ is called and depending on the result we either return Fail (in the case of (uni^-)) or return as value the value of P in context Γ after applying the unifier (uni^+).

The rule (MV_{intro}) is used to introduce a fresh meta-variable. When we need to evaluate a term of the form $?m : A.M$ in a state context Γ and local context Δ, we need to introduce a fresh meta-variable to the state context. This meta-variable however has a local context Δ that needs to be recorded. This explains why the state context in the premise of the rule is $\Gamma, ?m_1[\Delta] : A$. The local context is not changed, but we need to propagate the renaming of m in the body M of the tactic term.

The rule for evaluating defined tactics makes sure that the arguments of the tactic are well-formed. Then it propagates them through the body of the tactic and evaluates the result. We may think of T in this rule as one of the tactics defined in Section 3.

Example 4.1 Here are a few examples of derivable judgments:

$$\models \langle A : \mathsf{Prop}; \rangle \rhd \mathsf{Intros}[A \to A] \Rightarrow \langle A : \mathsf{Prop}, n[x : A] : A \rangle \rhd \lambda x : A.n[x] : A \to A$$
$$\models \langle A : \mathsf{Prop}; x : A \rangle \rhd \mathsf{Apply}[A, A, x] \Rightarrow \langle A : \mathsf{Prop}; x : A \rangle \rhd x : A$$
$$\models \langle U : \mathsf{Type}; t : U, f : U \to U \rangle \rhd ?m : U.[m \sim t](f\, m) \Rightarrow \langle \ldots \rangle \rhd (f\, t) : U$$
$$\models \langle A : \mathsf{Prop}, g : A \to A, ?m[\,] : \mathsf{Prop}; \rangle \rhd ([m[\,] \sim A].\lambda f : A \to A.f)(\lambda x : m[\,].gx) \Rightarrow$$
$$\Rightarrow \langle A : \mathsf{Prop}, g : A \to A, !m[\,] := A : \mathsf{Prop}; \rangle \rhd (\lambda f : A \to A.f)(\lambda x : m[\,].gx) : A \to A$$
$$\models \langle \Gamma_0; \rangle \rhd \mathsf{Apply}[(R\, a\, c), \forall xyz : U.(R\, x\, y) \to (R\, y\, z) \to (R\, x\, z), t] \Rightarrow$$
$$\Rightarrow \langle \Gamma_0, !x[\,] := a : U, ?y[\,] : U, !z[\,] := c : U, ?p[\,] : (R\, x[\,]\, y[\,]), ?q[\,] : (R\, y[\,]\, z[\,]) \rangle \rhd (t\, x[\,]\, y[\,]\, z[\,]\, p[\,]\, q[\,])$$

where Γ_0 is a context that declares the type U, elements a and c of U, the assumption t that R is transitive. For technical reasons it is more convenient not to unfold the definitions generated by a tactic although the unfolded version is more readable.

Next, we will show that the semantics $\models$ is sound and complete with respect to the typing relation of the object layer.

Proposition 4.2 *If* $\models \langle \Gamma; \Delta \rangle \rhd M \Rightarrow \langle \Gamma'; \Delta' \rangle \rhd N : A$ *then*

(i) $\Delta \equiv \Delta'$.

(ii) $\mathsf{dom}(\Gamma) \subseteq \mathsf{dom}(\Gamma')$.

Theorem 4.3 (Completeness) *Let* Γ *be a context and* Δ *a parameter list.*

If $\Gamma, \Delta \vdash M : A$ *then*

$$\models \langle \Gamma; \Delta \rangle \rhd M \Rightarrow \langle \Gamma; \Delta \rangle \rhd M : A$$

Proof. Induction on the derivation of $\Gamma, \Delta \vdash M : A$. □

Definition 4.4 [Γ-morphism] Let Γ be a context. The pair $\delta \equiv \langle \delta_\iota, \delta_\Sigma \rangle$ is called Γ-morphism if the following hold:

- δ_ι is a map from a subset of the meta-variables declared in Γ to the set of $\mathcal{B}$-terms.
- δ_Σ is a context containing no variable declarations (i.e. containing only meta-variable declarations and definitions).

The result of the application of a Γ-morphism δ to Γ is defined as the context $\delta_\iota(\Gamma), \delta_\Sigma$, where $\delta_\iota(\Gamma)$ is defined inductively on the initial segments of Γ:

$$\begin{aligned}
\delta_\iota(\varepsilon) &= \varepsilon \\
\delta_\iota(\Gamma', x:A) &= \delta_\iota(\Gamma'), x:A \\
\delta_\iota(\Gamma', ?m[\Delta]:A) &= \delta_\iota(\Gamma'), ?m[\Delta]:A && m \notin \mathsf{dom}(\delta_\iota) \\
\delta_\iota(\Gamma', ?m[\Delta]:A) &= \delta_\iota(\Gamma'), !m[\Delta] := \delta_i(m) : A && m \in \mathsf{dom}(\delta_\iota) \\
\delta_\iota(\Gamma', !m[\Delta] := t:A) &= \delta_\iota(\Gamma'), !m[\Delta] := t : A
\end{aligned}$$

Definition 4.5 Let Γ be a valid context. The Γ-morphism δ is called well-typed if $\delta(\Gamma)$ is a valid context.

Lemma 4.6 *Let* $\Gamma, \Delta \vdash M : A$ *and let* δ *be a well-typed* Γ*-morphism. Then*

$$\delta(\Gamma), \Delta \vdash M : A$$

Lemma 4.7 *Let* Γ *be a valid context. If* δ_1 *is a well-typed* Γ*-morphism and* δ_2 *is a well-typed* $\delta_1(\Gamma)$*-morphism, then the composition* $\delta_2 \circ \delta_1$ *is a well-typed* Γ*-morphism.*

Theorem 4.8 (Soundness) *Let* $\models \langle \Gamma; \Delta \rangle \rhd M \Rightarrow \langle \Gamma'; \Delta \rangle \rhd N : A$. *Then*

(i) Γ, Δ *is a valid context.*

(ii) *There is a well-typed* Γ*-morphism* δ *such that* $\delta(\Gamma) \equiv \Gamma'$.

(iii) $\Gamma', \Delta \vdash N : A$.

The soundness theorem states that the successfully evaluated tactic terms are well-typed in the calculus of open terms and gives us a relation between the context representing the state of the prover before and after the execution

of the tactic. This relation is expressed by the morphism δ. The existence of such a morphism shows that the final state is obtainable from the initial one by means of introduction and instantiation of meta-variables only.

As a side remark we could note that this is another example when we do need to distinguish between meta-variables and 'normal' variables.

5 Related and Future Work

5.1 Related Work

This paper builds on ideas already present in the literature. For example the representation of unknowns by parameterized meta-variables that we introduced in [3] can be seen as adding meta-information to the functional approach of Miller [6] where unknown terms are viewed as functions of the free variables occurring in them. Other ideas that have influenced this work come from systems like ALF where the first type-checking algorithm for open terms is presented; the $\Pi_{\mathcal{L}}$ calculus of Muñoz [7] where a calculus of open terms is combined with an explicit substitution calculus; the system TypeLab [9] where it is noticed that we only need to attach explicit substitutions to meta-variables; the system OLEG [5] where the binders for meta-variables inspired the introduction of the binder that we use to introduce fresh meta-variables. As mentioned already, from OLEG we also have taken the idea to use definitions instead of instantiations to solve meta-variables.

On the level of tactics we have the tactic language of Coq by Delahaye [2] where the basic tactics of Coq are taken as atomic actions and one can define complex tactics using different tacticals. One can see the present work as complementary because we focus our attention on a calculus that allows us to define these basic tactics in terms of simpler primitives and furthermore to explain their semantics in terms of the target object-level language.

Last, but not least, our work has been influenced by the Pure Pattern Type Systems (PPTSs) of Barthe et al. [1] which is an extension on previous work on the rewriting calculus (see e.g. [4]). Our approach relates to this work in the following way: In a PPTS one abstracts a pattern and then this pattern is matched to the actual argument given to the function. As in our case, this results in an instantiation. The fact that matching is performed (as opposed to unification) means that the pattern reduction is localized only to the redex that we contract, while in our case there may be side effects that make the reduction non-local.

5.2 Ongoing and Future Work

As we mentioned in the abstract, the work presented in this paper is still in progress. We still need to investigate whether the following conjecture holds

Conjecture 5.1 (Determinacy) *Let*

$$\models \langle \Gamma; \Delta \rangle \rhd M \Rightarrow \langle \Gamma'; \Delta \rangle \rhd N' : A'$$
$$\models \langle \Gamma; \Delta \rangle \rhd M \Rightarrow \langle \Gamma''; \Delta \rangle \rhd N'' : A''$$

Then $\Gamma' \equiv \Gamma''$, $N' \equiv N''$ *and* $\Gamma', \Delta \vdash A' =_{\beta\delta} A''$

and if not, under which restrictions on $uni(\,)$ it does.

Parallel to defining semantics for tactics we are working on a language of tacticals and its semantics. This is not a trivial extension of the current work because there are several specific problems that need to be addressed. Those include handling of the 'focus' (i.e. the current goal), issues with naming of the hypotheses, portability of tactical scripts, etc.

References

[1] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure pattern type systems. In G. Morrisett, editor, *POPL'03*. ACM Press, 2003.

[2] David Delahaye. A Tactic Language for the System Coq. In *Proceedings of Logic for Programming and Automated Reasoning (LPAR)*. Springer-Verlag LNCS/LNAI, November 2000.

[3] Herman Geuvers and Gueorgui Jojgov. Open Proofs and Open Terms: a Basis for Interactive Logic. In J. Bradfield, editor, *Proceedings of CSL'02*, number 2471 in LNCS, pages 537–552. Springer, 2002.

[4] Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. The Rho Cube. In Furio Honsell, editor, *Foundations of Software Science and Computation Structures,ETAPS'2001*, Lecture Notes in Computer Science, pages 166–180. Springer-Verlag, April 2001.

[5] Conor McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, University of Edinburgh, 1999.

[6] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 1992.

[7] César A. Muñoz. *A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory*. PhD thesis, INRIA, November 1997.

[8] R. Pollack. The LEGO Proof Assistant. http://www.dcs.ed.ac.uk/home/lego/index.html.

[9] M. Strecker. *Construction and Deduction in Type Theories*. PhD thesis, Universität Ulm, 1999.

[10] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V7.4*, February 2003. http://coq.inria.fr.

A The Calculus of Open Terms

$$\vdash \mathsf{Prop} : \mathsf{Type} \quad (ax_1) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \quad (\Pi)$$

$$\vdash \mathsf{Type} : \mathsf{Kind} \quad (ax_2) \qquad \frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B} \quad (\lambda)$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \quad (start) \qquad \frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]} \quad (app)$$

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash M : B}{\Gamma, x{:}A \vdash M : B} \quad (weak) \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s \quad \Gamma \vdash A = B}{\Gamma \vdash M : B} \quad (conv)$$

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash N_i : A_i[N_j/x_j]_{j<i}}{\Gamma \vdash m[\vec{N}] : A[\vec{N}/\vec{x}]} \quad (MV_{start}) \quad m[\vec{x}{:}\vec{A}] : A \in \mathsf{dom}(\Gamma)$$

$$\frac{\Gamma, \Theta \vdash A : s \quad \Gamma \vdash M : B}{\Gamma, ?m[\Theta]{:}A \vdash M : B} \quad (MV_{weak})$$

$$\frac{\Gamma, \Theta \vdash N : A : s \quad \Gamma \vdash M : B}{\Gamma, !m[\Theta] := N{:}A \vdash M : B} \quad (MD_{weak})$$

Typing of the terms of the object layer.

Lemma A.1 (Meta-Variable Weakening)

- *If* $\Gamma_1, \Gamma_2 \vdash M : B$ *and* $\Gamma_1, \Theta \vdash A : s$ *then* $\Gamma_1, ?m[\Theta]{:}A, \Gamma_2 \vdash M : B$
- *If* $\Gamma_1, \Gamma_2 \vdash M : B$ *and* $\Gamma_1, \Theta \vdash N : A : s$ *then* $\Gamma_1, !m[\Theta] := N{:}A, \Gamma_2 \vdash M : B$

where m *is a fresh meta-variable name.*

Lemma A.2 (Cut Lemma) *If* $\Gamma_1, ?m[\Theta]{:}A, \Gamma_2 \vdash M : B$ *and* $\Gamma_1, \Theta \vdash N : A$ *then*

- $\Gamma_1, \Gamma_2\{m[\Theta] := N\} \vdash M\{m[\Theta] := N\} : B\{m[\Theta] := N\}$
- $\Gamma_1, !m[\Theta] := N{:}A, \Gamma_2 \vdash M : B$

B Semantics rules for non-tactics term constructors

$$\models \langle\rangle \rhd \mathsf{Prop} \Rightarrow \langle\rangle \rhd \mathsf{Prop} : \mathsf{Type} \qquad (ax_1)$$

$$\models \langle\rangle \rhd \mathsf{Type} \Rightarrow \langle\rangle \rhd \mathsf{Type} : \mathsf{Kind} \qquad (ax_2)$$

$$\frac{\models \langle\Gamma;\Delta\rangle \rhd A \Rightarrow \langle\Gamma;\Delta\rangle \rhd A : s}{\models \langle\Gamma;\Delta, x:A\rangle \rhd x \Rightarrow \langle\Gamma;\Delta, x:A\rangle \rhd x : A} \qquad (start)$$

$$\frac{\models \langle\Gamma;\varepsilon\rangle \rhd A \Rightarrow \langle\Gamma;\varepsilon\rangle \rhd A : s}{\models \langle\Gamma, x:A;\varepsilon\rangle \rhd x \Rightarrow \langle\Gamma, x:A;\varepsilon\rangle \rhd x : A} \qquad (start_\Gamma)$$

$$\frac{\begin{array}{c}\models \langle\Gamma;\Delta\rangle \rhd A \Rightarrow \langle\Gamma;\Delta\rangle \rhd A : s \\ \models \langle\Gamma;\Delta\rangle \rhd M \Rightarrow \langle\Gamma';\Delta\rangle \rhd N : B\end{array}}{\models \langle\Gamma;\Delta, x:A\rangle \rhd M \Rightarrow \langle\Gamma';\Delta, x:A\rangle \rhd N : B} \qquad (weak)$$

$$\frac{\begin{array}{c}\models \langle\Gamma;\varepsilon\rangle \rhd A \Rightarrow \langle\Gamma;\varepsilon\rangle \rhd A : s \\ \models \langle\Gamma;\varepsilon\rangle \rhd M \Rightarrow \langle\Gamma';\varepsilon\rangle \rhd N : B\end{array}}{\models \langle\Gamma, x:A;\varepsilon\rangle \rhd M \Rightarrow \langle\Gamma', x:A;\varepsilon\rangle \rhd N : B} \qquad (weak_\Gamma)$$

$$\frac{\begin{array}{c}\models \langle\Gamma;\Delta\rangle \rhd A \Rightarrow \langle\Gamma;\Delta\rangle \rhd A : s_1 \\ \models \langle\Gamma;\Delta, x:A\rangle \rhd B \Rightarrow \langle\Gamma';\Delta, x:A\rangle \rhd B' : s_2\end{array}}{\models \langle\Gamma;\Delta\rangle \rhd \Pi x:A.B \Rightarrow \langle\Gamma';\Delta\rangle \rhd \Pi x:A.B' : s_2} \qquad (\Pi)$$

$$\frac{\begin{array}{c}\models \langle\Gamma;\Delta, x:A\rangle \rhd M \Rightarrow \langle\Gamma';\Delta, x:A\rangle \rhd N : B \\ \models \langle\Gamma';\Delta\rangle \rhd \Pi x:A.B \Rightarrow \langle\Gamma';\Delta\rangle \rhd \Pi x:A.B : s\end{array}}{\models \langle\Gamma;\Delta\rangle \rhd \lambda x:A.M \Rightarrow \langle\Gamma';\Delta\rangle \rhd \lambda x:A.N : \Pi x:A.B} \qquad (\lambda)$$

$$\frac{\begin{array}{c}\models \langle\Gamma;\Delta\rangle \rhd M \Rightarrow \langle\Gamma';\Delta\rangle \rhd P : \Pi x:A.B \\ \models \langle\Gamma';\Delta\rangle \rhd N \Rightarrow \langle\Gamma'';\Delta\rangle \rhd Q : A\end{array}}{\models \langle\Gamma;\Delta\rangle \rhd MN \Rightarrow \langle\Gamma'';\Delta\rangle \rhd PQ : B[Q/x]} \qquad (app)$$

,

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd M \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : A \quad \models \langle \Gamma';\Delta\rangle \rhd B \Rightarrow \langle \Gamma';\Delta\rangle \rhd B : s \quad \Gamma',\Delta \vdash A =_{\beta\delta} B}{\models \langle \Gamma;\Delta\rangle \rhd M \Rightarrow \langle \Gamma';\Delta\rangle \rhd N : B}\ (conv)$$

$$\frac{\models \langle \Gamma;\Delta\rangle \rhd A \Rightarrow \langle \Gamma;\Delta\rangle \rhd A : s \quad \models \langle \Gamma;\Delta\rangle \rhd N_i \Rightarrow \langle \Gamma;\Delta\rangle \rhd N_i : A_i[N_j/x_j]_{j<i}}{\models \langle \Gamma;\Delta\rangle \rhd m[\vec{N}] \Rightarrow \langle \Gamma;\Delta\rangle \rhd m[\vec{N}] : A[\vec{N}/\vec{x}]}\ (MV_{start})\ m[\vec{x}:\vec{A}]:A \in \mathsf{dom}(\Gamma)$$

$$\frac{\models \langle \Gamma;\Theta\rangle \rhd A \Rightarrow \langle \Gamma;\Theta\rangle \rhd A : s \quad \models \langle \Gamma;\varepsilon\rangle \rhd M \Rightarrow \langle \Gamma';\varepsilon\rangle \rhd N : B}{\models \langle \Gamma, ?m[\Theta]:A;\varepsilon\rangle \rhd M \Rightarrow \langle \Gamma', ?m[\Theta]:A;\varepsilon\rangle \rhd N : B}\ (MV_{weak})$$

$$\frac{\models \langle \Gamma;\Theta\rangle \rhd t \Rightarrow \langle \Gamma;\Theta\rangle \rhd t : A \quad \models \langle \Gamma;\Theta\rangle \rhd A \Rightarrow \langle \Gamma;\Theta\rangle \rhd A : s \quad \models \langle \Gamma;\varepsilon\rangle \rhd M \Rightarrow \langle \Gamma';\varepsilon\rangle \rhd N : B}{\models \langle \Gamma, !m[\Theta]:=t:A;\varepsilon\rangle \rhd M \Rightarrow \langle \Gamma', !m[\Theta]:=t:A;\varepsilon\rangle \rhd N : B}\ (MD_{weak})$$