Karnak* an automated theorem prover for PPC*

Tarek Mohamed Elnadi   Albert Hoogewijs

Abstract

In this paper we introduce KARNAK*, an automated theorem prover for the partial predicate calculus PPC*. PPC* has been introduced in [8] as an equivalent logic for LPF, the Logic of Partial Functions which is the logical basis of the software specification language VDM [10]. KARNAK* is used to show that a complete subsystem of LPF is derivable from PPC*, and hence it follows that PPC* is also complete. In addition, theorem preserving transformations between PPC and PPC* are introduced.

1 Introduction

We tried to make the paper as self-contained as possible. It is organized as follows. In section 2, the partial predicate calculi PPC, LPF and PPC* are discussed and the deduction rules of PPC and LPF are presented in Appendix A and Appendix B respectively. The automated theorem prover KARNAK* is explained in section 3, and used in section 4 to prove the completeness of PPC*. Some of the completeness proofs are introduced in Appendix C. Theorem preserving transformations between PPC and PPC* are introduced in section 5, while section 6 contains some concluding remarks.

2 The Partial Predicate Calculi PPC, LPF and PPC*

In [7] the partial predicate calculus PPC was introduced in order to formalize the undefinedness notion both in mathematics and in computer science. The basic

Received by the editors November 1994 - Revised February 1995
Communicated by Y. Félix

propositional connectives are $\neg$ and $\wedge$. These connectives together with the classical definitions of $\vee, \rightarrow, \leftrightarrow$ satisfy the Kleene K_3 truth tables [13]. In addition, the Δ connective is used and corresponds to Hallden's "meaningful"-functor, where $\Delta\alpha$ means that α is defined. The $\forall$ quantifier and the derived $\exists$ quantifier correspond to their definitions in the finite Łukasiewicz and Post logics where for any valuation $\mathcal{V}$ in a domain $\mathcal{D}$, $\mathcal{V}(\forall x\alpha(x)) \equiv \min\{\mathcal{V}(\alpha(a)) : a \in \mathcal{D}\}$ ($\mathcal{V}(\exists x\alpha(x)) \equiv \max\{\mathcal{V}(\alpha(a)) : a \in \mathcal{D}\}$). The validity notion used to define $\Gamma \models \alpha$ is different from the classical approach, in the sense that $\Gamma \models \alpha$ means: for all models if all the formulas of Γ are evaluated to "true", then the formula α is NOT evaluated to "false". It follows that the classical propositional part of PPC corresponds to the following "Kleene matrix":

$$K_3' = (\{F, U, T\}, \neg, \rightarrow, \vee, \wedge, \leftrightarrow \{U, T\})$$

where $\{U, T\}$ refers to the set of designated values [13]. It has been shown that this matrix is homomorphic to the classical two-valued matrix, in the sense that the set of tautologies coincides with that of the classical logic [2].

According to the terminology of Ryan and Sadler in [14], PPC was formalized as a "natural deduction calculus in sequent style" (see Appendix A). This means that instead of manipulating formulas as in natural deduction, one manipulates whole sequents. However, the set on the right of the turnstile can contain only one formula[1]. While pure sequent calculi only contain introduction rules and hence satisfy the "subformula property", this natural sequent calculus contains both introduction and elimination rules expressing the basic properties of the connectives and quantifiers. In [6, 9] KARNAK was introduced as an ATP for PPC and it was explained how to overcome the problems that arise from this feature of PPC.

In [1, 4] LPF was introduced as a logic covering undefinedness in program proofs, and the deduction rules of LPF are presented in Appendix B. The basic language of LPF consists of the connectives $\neg$ and $\vee$ which satisfy the K_3 tables, the Δ connective and the $\exists$ quantifier, but also the propositional symbol "*uu*" which is always undefined and which was introduced by Słupecki in 1936 in order to make the three-valued calculus of Łukasiewicz (Ł$_3$) functionally complete [2]. With respect to the model theory, $\Gamma \models_{\mathbf{LPF}} \alpha$ has the classical meaning in the sense that for all models if all the formulas of Γ are evaluated to "true" then the formula α is evaluated to "true". In this case, the classical fragment of the propositional logic has a void set of tautologies since it corresponds to the classical K_3 matrix

$$K_3 = (\{F, U, T\}, \neg, \rightarrow, \vee, \wedge, \leftrightarrow \{T\})$$

(see [2]). In [4] the notation $\alpha \supset \beta$ is used as an abbreviation for $\neg\alpha \vee \neg\Delta\alpha \vee \beta$ in order to obtain a Hilbert-style axiomatization of the propositional part of LPF.

In [8] PPC* was introduced as a modified version of PPC having the validity notion of LPF. The PPC* rules were obtained from the PPC rules by removing the rules (E), (D$_1$) and (D$_2$) and adding the following new rules:

$$(\mathrm{E}') \quad \neg(t = t) \vdash^\star \alpha \qquad (\Delta) \quad \frac{\Gamma \ \vdash^\star \ \alpha}{\Gamma \ \vdash^\star \ \Delta\alpha}$$

[1]This is not a real restriction since one can present a list of formulas as its disjunction.

$$(\mathrm{Cn}_0)\quad \frac{\begin{array}{ll}\Gamma_1, \neg\alpha_1 & \vdash^\star \beta \\ \Gamma_2, \neg\alpha_2 & \vdash^\star \beta\end{array}}{\Gamma_1, \Gamma_2, \neg(\alpha_1 \wedge \alpha_2) \vdash^\star \beta}$$

$$(\mathrm{Gn}_x)\quad \frac{\Gamma, \neg\alpha \vdash^\star \beta}{\Gamma, \neg\forall x\alpha \vdash^\star \beta} \qquad \text{if } x \text{ is not free in } \Gamma, \beta$$

The rule (Δ) was added to realize the validity notion of LPF. The rules (D_1) and (D_2) were removed because either of them with (Δ) imply that all formulas are defined. In [8] it was shown that in PPC (E) is equivalent to (E$'$). Since (E) with (Δ) imply that all terms are defined, (E) was replaced by (E$'$). Since the contraposition rule (CaPo_1) $\left(\dfrac{\Gamma, \alpha \vdash \beta}{\Gamma, \neg\beta \vdash \neg\alpha}\right)$ and (Ad_1) imply (D_2), (CaPo_1) is not sound and the existing PPC proofs of (Cn_0) and (Gn_x) which rely on (CaPo_1) are no longer valid. Hence, (Cn_0) and (Gn_x) were added seeking for the completeness.

Since PPC* was obtained from PPC by removing some rules and adding other ones, the system has some redundancies:

- The rule (X$'$) can be replaced by its classical equivalent (X), and the second assumption of the rule (Cn_i) is not needed anymore.
- The cut rule is provable in PPC*, and hence a rule like (C_0) becomes equivalent to its abbreviated form $\alpha, \beta \vdash^\star \alpha \wedge \beta$.
- Some rules are obviously dependent, especially due to the addition of the rule (Δ). For example, (Ad_1) can be derived from (A) and (Δ), and (Cd_i) can be derived from (C_i) and (Δ).

The soundness of PPC* can easily be derived by showing that each of its rules is sound. For example, to prove the soundness of the substitution rule (S'^{t}_{x}) we show the following:

$$(\mathrm{S}'^{t}_{x})\quad \frac{\begin{array}{ll}\Gamma_1 & \models^\star \alpha \\ \Gamma_2 & \models^\star \Delta(t = t)\end{array}}{\Gamma_2, \Gamma_3 \models^\star \beta} \qquad \text{if } \mathbf{subst}xt\Gamma_1\Gamma_3 \text{ and } \mathbf{subst}xt\alpha\beta$$

Assume $\mathbf{subst}xt\Gamma_1\Gamma_3$, $\mathbf{subst}xt\alpha\beta$, $\Gamma_1 \models^\star \alpha$ and $\Gamma_2 \models^\star \Delta(t = t)$. Assume also that $\mathcal{M} \equiv (\mathcal{D}, \mathcal{I})$ is a PPC* model satisfying Γ_2, Γ_3 in the sense that all the formulas of Γ_2 and Γ_3 are true in $\mathcal{M}$. Then, one has to prove that β is also true in $\mathcal{M}$. Since $\Gamma_2 \models^\star \Delta(t = t)$ and $\mathcal{M}$ satisfies Γ_2, $\mathcal{V}\Delta(t = t) \equiv T$ which means that the term t is defined in $\mathcal{M}$ and hence there is an interpretation $\mathcal{I}t \in \mathcal{D}$. Now consider the model $(\mathcal{D}, \mathcal{I}_x^{\mathcal{I}t})$ where $\mathcal{I}_x^{\mathcal{I}t}$ is exactly like $\mathcal{I}$ but x is interpreted as $\mathcal{I}t$. Since $\mathcal{M} \equiv (\mathcal{D}, \mathcal{I})$ satisfies Γ_3 and $\mathbf{subst}xt\Gamma_1\Gamma_3$, $(\mathcal{D}, \mathcal{I}_x^{\mathcal{I}t})$ satisfies Γ_1. Then, $(\mathcal{D}, \mathcal{I}_x^{\mathcal{I}t})$ also satisfies α because $\Gamma_1 \models^\star \alpha$. Hence, $\mathcal{M} \equiv (\mathcal{D}, \mathcal{I})$ satisfies β because $\mathbf{subst}xt\alpha\beta$. $\square$

The soundness of the other rules can be proven similarly.

As will be proven in section 4, PPC* is complete. In addition, it is equivalent to LPF$'$ which is a logic obtained from LPF by excluding the symbol "*uu*" and its rules.

3 KARNAK*

KARNAK* is a modified version of KARNAK. The proof development method (PDM) and the proof strategy of KARNAK* are similar to those of KARNAK which are described in detail in [6, 9].

It was already noted that PPC and PPC* are natural sequent calculi and not pure sequent calculi for which the backwards PDM can be implemented efficiently [3, 12, 15] due to the subformula property. The forwards application of introduction rules and the backwards application of elimination rules are explosive. This means that neither the forwards nor the backwards PDM can easily be used with natural calculi. The PDM of KARNAK and KARNAK* is mainly forwards. The program takes a set of assumptions and a conjecture and tries to prove the conjecture by successive applications of the deduction rules on the given assumptions. In addition, a backwards technique is used to look ahead for the conjecture.

The proof strategies used with natural systems are generally intended to capture some aspect of human reasoning and they should not necessarily be complete [5]. If *generality* is defined as the degree of completeness of some proof strategy, the designer of some ATP should look for an intelligent strategy satisfying the balance between generality and efficiency. The proof strategy of KARNAK and KARNAK* is based on the following:

1. **Using filters to avoid to add steps which seem to be useless:**

   When the program succeeds to apply some rule on previous steps, it performs certain checks on the deduced step before adding it to the proof. The predicates which perform these checks are called *filters*. Then, the role of the filters comes after the application of rules, and they are proposed to make a distinction between the steps which seem to be useful and the ones which seem to be useless. The first filter ensures that the deduced step does not exist in the proof. The other filters are designed to avoid the steps which can be deduced directly from one of the rules (A), (Ad$_1$), (Ad$_2$), (D$_0$), (Cd$_1$), (Cd$_2$), (Gd$_1$) and (Gd$_3$), but were deduced using one of the other rules. It seems useless for example to derive the sequent $\alpha \vdash \alpha$ using a rule different from (A). These filters do not reduce much the generality of the program but may increase its efficiency.

2. **Using a technique to avoid to repeat the previous computations:**

   Suppose that the program is trying to prove some theorem and has generated one hundred steps. If the program fails to apply some rule like (R$'$) on the hundred steps to add a new step to the proof, this fact is registered in a certain database so that the program avoids in the following proving procedure to choose the three sequents, on which the rule (R′) is to be applied, from the first hundred steps. This helps to save much time especially in long proofs, and hence increases the efficiency of the program.

3. **Restricting the application of some rules:**

   Some of the PPC deduction rules have the property that the conclusion of the rule is not determined completely by its assumptions. These rules are (A), (Ad_1), (Ad_2), (Cd_1), (Cd_2), (Cn_1), (Cn_2), (D_0), (X′), (Gd_1), (Gd_3), (E), (Ed_1), (Ed_2) and also (De_1) in some sense. In order to apply these rules, one should specify the unknown parts in the conclusion. For example, to apply some rule like (A) $\alpha \vdash \alpha$, one should specify which α must be used. Since one can choose any formula α, the uncontrolled application of such rule is explosive and may generate an enormous number of sequents. Hence, it is unavoidable to restrict its application. In addition, a rule like (C_0) can be applied on any two previous sequents producing the same problem although the conclusion of this rule is completely determined by its assumptions. Then, the other rules may also be explosive and one should put restrictions on their application. The problem is how to restrict the application of the explosive rules without reducing much the generality of the program. Due to these restrictions, the program will not be able to discover the proofs beyond them. For example, the program does not use any conjunction which does not exist in the theorem. Hence, it may not be able to discover a proof for the following theorem:

$$\text{(AnEx)} \quad \frac{\Gamma \vdash \alpha}{\Gamma, \beta \vdash \alpha}$$

   which is proven in [7] as follows:

1:	Γ	$\vdash$	α	assumption
2:	β	$\vdash$	β	(A)
3:	Γ, β	$\vdash$	$\alpha \wedge \beta$	(C_0) on 1,2
4:	Γ, β	$\vdash$	α	(C_1) on 3 $\square$

   These rule and proof are valid in PPC and PPC*.

   In the following, we introduce the principles which are used in KARNAK* to restrict and control the application of the deduction rules. These principles are similar to those of KARNAK with slight modifications.

   Principle 1:
   The program does not use any conjunction which does not exist in the theorem. This principle is used to control the rules (C_0), (Cd_0), (Cd_1), (Cd_2), (Cn_1), (Cn_2) and (Cn_0).

   Principle 2:
   The rules (G_x), (Gd_1), (Gd_3) and (Gn_x) are controlled by using only the quantifications which are in the theorem. One can notice that if x does not

occur free in Γ in a sequent $\Gamma \vdash^\star \alpha$, then (G_x) can be applied on this sequent to give $\Gamma \vdash^\star \forall x\alpha$. Since x does not occur free in Γ, (G_x) can be applied again on $\Gamma \vdash^\star \forall x\alpha$ to give $\Gamma \vdash^\star \forall x\forall x\alpha$. This may cause infinite useless computations. "Principle 2" controls the application of (G_x) where any quantification will not be used if it is not in the theorem. One can also notice that the rule (Gd_2) can not be explosive.

Principle 3:
When some rule can be applied only in a small finite number of ways, the program uses all these ways without restrictions. This principle is used to control (De_1).

Principle 4:
The program does not use any variable or constant which does not exist in the theorem. This principle is used to control (Ed_1), (Ed_2) and (S'^t_x).

Principle 5:
If $\mathcal{F}$ is the set of the formulas of the theorem and their subformulas, the program applies the rules (Ad_1), (Ad_2) and (D_0) for $\alpha \in \mathcal{F}$. The rule (X') is implemented as its classical equivalent (X), and the program applies it for $\beta \in \mathcal{F}$. In addition, the program applies the rule (A) for $\alpha \in \mathcal{F} \cup \neg(\mathcal{F}) \cup \neg\Delta(\mathcal{F})$ where $\neg(\mathcal{F}) = \{\neg\alpha : \alpha \in \mathcal{F}\}$ and $\neg\Delta(\mathcal{F}) = \{\neg\Delta\alpha : \alpha \in \mathcal{F}\}$. This restriction reflects the principle of the "excluded fourth". We notice that many proofs are "normal" according to these restrictions. In other words, this principle may not reduce much the generality of the program.

Principle 6:
The program applies the rule (Dn_1) on a sequent $\Gamma \vdash^\star \Delta\alpha$ only when $\alpha \in \mathcal{F}$. Without such restriction, (Dn_1) can be applied an infinite number of times, where it can be applied on $\Gamma \vdash^\star \Delta\alpha$ to give $\Gamma \vdash^\star \Delta\neg\alpha$, then it can be applied on $\Gamma \vdash^\star \Delta\neg\alpha$ to give $\Gamma \vdash^\star \Delta\neg\neg\alpha$, and so on. For a similar reason, the rule (Δ) is applied only for $\alpha \in \mathcal{F}$.

Principle 7:
The rule (E') is implemented in another equivalent and more deterministic form $\neg(t = t) \vdash^\star t = t$. This rule becomes explosive when the theorem contains a function symbol because in this case the universe of discourse is infinite. If $\mathcal{T}$ is the set of the terms of the theorem and their subterms, the program applies the rule (E') for $t \in \mathcal{T}$. However, the user can overcome this restriction by introducing terms outside $\mathcal{T}$ as shown in [6, 9].

Principle 8:
The program applies the rule (E^t_x) for $(x = t) \in \mathcal{F}$.

By experiment, it was noticed that there is some kind of inverse proportion between the efficiency and the generality of the program. In other words, when one decreases the restrictions on the application of the rules to make the program more general, it is likely to be less efficient. It is really difficult to find a proof strategy which works efficiently for a wide range of problems. However, the current implementaion succeeded to prove many theorems in short times, and this can be considered a hopeful start. If one can add new rules to the program (regardless if it succeeded to prove them or not), it may be able to discover the proofs of much more theorems. Nevertheless, in order to add new rules to the program, the user should control the application of these rules, and this requires a good understanding of the program and the used programming language. The addition of new rules can hardly be automated but may be described systematically in a user's manual. By adding new rules, one may build theories of partial predicates and functions. Although the cut rule is provable in PPC*, it is not implemented in KARNAK* as a derived rule because it seems to decrease the efficiency of the program by generating many useless steps. However, the program is able to prove this rule in a short time and the proof is introduced in Appendix D.

When the program succeeds to prove some theorem, it can write a report file containing some statistics of the proof. The report contains the following information:

- The *Proving Time PT* in seconds, which is the time taken to prove the theorem.
- The *Writing Time WT* in seconds, which is the time taken to write the proof in LaTeX. Before writing the proof, the program removes all the unused steps from the proof with renumbering the reference numbers in the subsequent steps[2]. Hence, the proofs written by the program are completely connected. This may also help to reduce the assumptions because the unnecessary ones may not appear in the written proof. The time taken to remove the unused steps is included in *WT*.
- The *Derivation Length DL*, which is the number of steps generated for proving the theorem, including the unused ones.
- The *Proof Length PL*, which is the number of steps in the written proof.
- The *COnnectivity CO*, which is defined by the equation $CO = 100 \times PL/DL$. It measures the connectivity of the generated proof.
- The *DEpendency DE*, which is defined by the equation

  $$DE = \frac{1 + \sum_{i=1}^{PL} |L_i|}{PL}$$

  where L_i is the reference list of the ith step in the written proof. If the step is an assumption or deduced by one of the rules (A), (Ad$_1$), (Ad$_2$), (D$_0$), (E$'$), (Ed$_1$), (Ed$_2$), (Cd$_1$), (Cd$_2$), (Gd$_1$) and (Gd$_3$), then its reference list is taken to be the empty list. One can notice that $|L_1| = 0$ for any proof. However, "$i = 1$" is included in the sum to cover the case in which $PL = 1$, for example when the theorem is $\alpha \vdash^\star \alpha$. Since $\sum_{i=1}^{PL} |L_i| = \sum_{i=1}^{PL} n_i$ where n_i is the number

[2]Some step is unused iff it is not the last step and its number does not appear in any reference list in the subsequent steps.

of times the ith step is used, DE can be taken as a measure of the dependency of the proof steps, because it is the average number of times for using each step of the proof. The idea behind adding "1" in the numerator of the equation, is that one can imagine an additional step from the conjecture to the sign $\Box$ with the reference list $\{PL\}$. Since the written proofs are completely connected, $DE \geq 1$ for any proof.

- The *Deduction Speed DS*, which is defined by the equation $DS = PL/PT$.
- The *Step Time ST*, which is defined by the equation $ST = 1/DS$.

PT, WT, DS and ST are machine-dependent, while the other parameters are machine-independent. When CO and DS increase, and PT and DL decrease, this denotes that the proof strategy is more efficient towards the considered theorem. The reports provide some kind of *self-evaluation* of the program and help much to evaluate any modification in it. This is especially useful in the development stage. In addition, a statistical study on the relationship between the different parameters of the reports, may provide a good insight for the characteristics of the used proof strategy. The usefulness of an ATP like KARNAK* is a relative matter and can be measured by comparison between the abilities of a human and the ATP to prove theorems, and such comparison can be made using the reports.

KARNAK and KARNAK* are implemented in "PDC prolog 3.3" and have been developed and tested on an IBM 486DX personal computer with 8 MB RAM and frequency 66 MHz. To prove some theorem using the program, the user first writes the theorem in KTS (KARNAK Theorem Syntax) which is a prefix language compatible with PDC prolog. Then, the user runs the program to prove the theorem. For more details about how to use the program, we refer to [9]. In the appendices of this paper, some of the proofs generated by the program are introduced. In each example, the arrow $\longrightarrow$ marks the KTS input, $\Leftarrow$ marks the LaTeX proof which is automatically written by the program, and $\longleftarrow$ marks the report. We emphasize that the LaTeX proofs introduced after the arrow $\Leftarrow$ are written by the program without any human modification.

4 Using KARNAK* to Prove the Completeness of PPC*

LPF was proven to be complete in [4] where it was first formalized as a complete sequent calculus S-LPF. Then, a natural deduction style N-LPF was introduced and proven to be complete using the completeness of S-LPF. The deduction rules of both N-LPF and S-LPF are presented in Appendix B. Assume that N-LPF$'$ is the calculus obtained from N-LPF by excluding the symbol "*uu*" and its rules "*uu-E*" and " $\neg$*uu-E*", and S-LPF$'$ is the calculus obtained from S-LPF by excluding the symbol "*uu*" and its rules (3) and (4). The completeness of PPC* will be proven as follows: first, the completeness of S-LPF$'$ is derived from the completeness of S-LPF. Second, the completeness of N-LPF$'$ is derived from that of S-LPF$'$. Third, KARNAK* is used to derive all the N-LPF$'$ rules in PPC*.

Theorem 4.1 *S-LPF′ is complete.*

proof.

Assume $\models_{\mathbf{S\text{-}LPF'}} \Gamma \longrightarrow \Sigma$. Then, $\models_{\mathbf{S\text{-}LPF}} \Gamma \longrightarrow \Sigma$ which leads to $\vdash_{\mathbf{S\text{-}LPF}} \Gamma \longrightarrow \Sigma$ by the completeness of S-LPF. One can notice that each of the S-LPF rules presented in Appendix B has the property that if "*uu*" does not exist in the conclusion, then "*uu*" can not exist in the assumptions. For example, in the rule (8) if $\Delta\alpha$ is *uu*-free (in the sense that it does not contain "*uu*") then both α and $\neg\alpha$ are *uu*-free. Since Γ and Σ are S-LPF′ lists of formulas, they are *uu*-free. Then, the sequent $\Gamma \longrightarrow \Sigma$ is also *uu*-free. Then, the S-proof tree of $\vdash_{\mathbf{S\text{-}LPF}} \Gamma \longrightarrow \Sigma$ is *uu*-free. Hence, one has $\vdash_{\mathbf{S\text{-}LPF'}} \Gamma \longrightarrow \Sigma$. □

Lemma 4.2 *If* $\vdash_{\mathbf{S\text{-}LPF'}} \Gamma \longrightarrow \Sigma$, *then* $\Gamma \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$
where $\overline{\Sigma}$ *is the disjunction of the formulas of* Σ *if* Σ *is non-empty, or the formula* $\neg(c = c)$ *if* Σ *is empty. c is any constant symbol.*

proof.

The proof goes by induction on the number of sub-S-proofs[3] of $\vdash_{\mathbf{S\text{-}LPF'}} \Gamma \longrightarrow \Sigma$. If there is no sub-S-proofs, then there is only one sequent in the S-proof tree, and this sequent must match one of the rules (1), (2), (5), (6), (7), (24), (25), (26), (27), (28) and (29), and in each case, one can derive $\Gamma \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$ directly from "$\alpha \vdash_{\mathbf{N\text{-}LPF'}} \alpha$", (*contr*), (*def-c*), (*def-v*), (¬ *E-E*), (=-*subst2*), (=-*subst1*), (=-*reflx*), (¬ =-*reflx-L*), (¬ =-*reflx-R*) and (=-*2-val*) respectively.

To prove the induction step, assume that the S-proof has at least one sub-S-proof. Then, the last inference of the S-proof can be either of the following two forms:

$$\frac{\Gamma_1 \longrightarrow \Sigma_1 \qquad \Gamma_2 \longrightarrow \Sigma_2}{\Gamma \longrightarrow \Sigma} \qquad\qquad \frac{\Gamma_1 \longrightarrow \Sigma_1}{\Gamma \longrightarrow \Sigma}$$

Then, it is sufficient to show that for each of the S-LPF′ rules (8)-(23), one can prove $\Gamma \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$ from $\Gamma_1 \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma_1}$ [and $\Gamma_2 \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma_2}$] (the square brackets mean that the enclosed text is optional). Here, we prove that for (9).

Assume

Γ
$\neg\alpha, \Gamma \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$
$\alpha, \Gamma \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$

Then, one has to prove $\overline{\Sigma, \neg\Delta\alpha}$. From the assumptions, one gets $\neg\alpha \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$ and $\alpha \vdash_{\mathbf{N\text{-}LPF'}} \overline{\Sigma}$. If Σ is empty, then the formulas $\overline{\Sigma}$ and $\overline{\Sigma, \neg\Delta\alpha}$ are $\neg(c = c)$ and $\neg\Delta\alpha$ respectively, and the proof comes by

[3]Given a S-proof P, if P contains only one sequent, then P has no sub-S-proofs. If P contains more than one sequent, then its sub-S-proofs are the S-proof(s) P_1 (and P_2) obtained by removing the end sequent from P, plus the sub-S-proofs of P_1 (and P_2).

$$\frac{\neg\alpha \vdash_{\textbf{N-LPF}'} \neg(c=c);\ \alpha \vdash_{\textbf{N-LPF}'} \neg(c=c)}{\neg\Delta\alpha}$$

which is proven as follows:
1: $\neg\alpha \vdash \neg(c=c)$ assumption
2: $\alpha \vdash \neg(c=c)$ assumption
3: $\Delta\alpha \vdash \Delta\alpha$
4: $\Delta\alpha \vdash \neg\Delta\alpha$

4.1: $\Delta\alpha$ premise
4.2: $\neg(c=c)$ (Δ-*E*) on 4.1, 2, 1
4.3: $\neg\Delta\alpha$ ($\neg$ *E*-*E*) on 4.2

5: $\neg\Delta\alpha$ ($\neg\Delta$-*I*) on 3, 4 □

In the previous proof, the temporary assumptions are called "premises" while the permanent assumptions are called "assumptions".

If Σ is non-empty, then the formula $\overline{\Sigma, \neg\Delta\alpha}$ is $\overline{\Sigma} \vee \neg\Delta\alpha$, and the proof comes by

$$\frac{\neg\alpha \vdash_{\textbf{N-LPF}'} \overline{\Sigma};\ \alpha \vdash_{\textbf{N-LPF}'} \overline{\Sigma}}{\overline{\Sigma} \vee \neg\Delta\alpha}$$

which is proven as follows:
1: $\neg\Delta\Delta\alpha \vdash \Delta\Delta\alpha$

1.1: $\neg\Delta\Delta\alpha$ premise
1.2: $\neg\Delta\alpha \vdash \neg\Delta\Delta\alpha$

1.2.1: $\neg\Delta\Delta\alpha$ 1.1

1.3: $\neg\Delta\alpha \vdash \Delta\Delta\alpha$

1.3.1: $\neg\Delta\alpha$ premise
1.3.2: $\Delta\Delta\alpha$ (Δ-*I*-*2*) on 1.3.1

1.4: $\Delta\alpha$ ($\neg\Delta$-*E*) on 1.3, 1.2
1.5: $\Delta\Delta\alpha$ (Δ-*I*-*1*) on 1.4

2: $\neg\Delta\Delta\alpha \vdash \neg\Delta\Delta\alpha$
3: $\Delta\Delta\alpha$ ($\neg\Delta$-*E*) on 1, 2
4: $\alpha \vdash \overline{\Sigma}$ assumption
5: $\neg\alpha \vdash \overline{\Sigma}$ assumption
6: $\Delta\alpha \vdash \overline{\Sigma} \vee \neg\Delta\alpha$

6.1: $\Delta\alpha$ premise
6.2: $\overline{\Sigma}$ (Δ-*E*) on 6.1, 4, 5
6.3: $\overline{\Sigma} \vee \neg\Delta\alpha$ ($\vee$-*I*-*L*) on 6.2

7: $\neg\Delta\alpha \vdash \overline{\Sigma} \vee \neg\Delta\alpha$

7.1: $\neg\Delta\alpha$ premise
7.2: $\overline{\Sigma} \vee \neg\Delta\alpha$ ($\vee$-*I*-*R*) on 7.1

8: $\overline{\Sigma} \vee \neg\Delta\alpha$ (Δ-*E*) on 3, 6, 7 □
The other cases are similar. □

Theorem 4.3 *N-LPF′ is complete.*

proof.
Assume $\Gamma \models_{\textbf{N-LPF}'} \alpha$. Then, one has $\models_{\textbf{S-LPF}'} \Gamma \longrightarrow \alpha$ which leads to $\vdash_{\textbf{S-LPF}'} \Gamma \longrightarrow \alpha$ by theorem 4.1. Then, one has $\Gamma \vdash_{\textbf{N-LPF}'} \alpha$ by lemma 4.2. □

Theorem 4.4 $\Gamma \vdash_{\textbf{N-LPF}'} \alpha \Rightarrow \Gamma \vdash^\star \alpha$

proof.
KARNAK* has been used to derive all the N-LPF′ rules, and some of these proofs are introduced in Appendix C. □

Corollary 4.5 *PPC** *is complete.*

proof.
Assume $\Gamma \models^\star \alpha$. Since PPC* and N-LPF′ have the same model theory, $\Gamma \models_{\textbf{N-LPF}'} \alpha$ which leads to $\Gamma \vdash_{\textbf{N-LPF}'} \alpha$ by theorem 4.3. Then, one has $\Gamma \vdash^\star \alpha$ by theorem 4.4. □

Corollary 4.6 $\Gamma \vdash_{\textbf{N-LPF}'} \alpha \Leftrightarrow \Gamma \vdash^\star \alpha$

proof.
The right arrow follows from theorem 4.4, while the left arrow follows from the soundness of PPC* and theorem 4.3. □ This corollary means that PPC* is equivalent to N-LPF′ and hence to N-LPF if the symbol “*uu*” and its rules are excluded.

The PPC* rules (Gd$_1$) and (Gd$_2$) were not used in any of the KARNAK* proofs for the N-LPF′ deduction rules. This means that these rules can be removed from PPC* without affecting its completeness. Hence, they must be dependent on the other ones. The rule (Gd$_1$) can easily be derived from (G) and (Δ). When the rule (Gd$_2$) was removed from KARNAK*, the program has succeeded to prove it and the proof is introduced in Appendix D. However, it may be useful to keep these rules in KARNAK* because they may help to discover more proofs. This is also the case with respect to the other dependent rules such as (Ad$_1$) and (Cd$_i$).

5 Theorem Preserving Transformations between PPC and PPC*

Theorem 5.1 *If* Γ *is a list of PPC formulas and* α *is a PPC formula, then*

$$\Gamma \vdash \alpha \textit{ iff } \Gamma \vdash^\star \neg(\neg\alpha \wedge \Delta\alpha)$$
and
$$\Gamma \vdash^\star \alpha \textit{ iff } \Gamma \vdash \alpha \wedge \Delta\alpha$$

proof.

The proof follows directly from the adequacy of PPC and PPC* and from the equivalences

$$\begin{gathered}\Gamma \models \alpha \text{ iff } \Gamma \models^\star \neg(\neg\alpha \wedge \Delta\alpha)\\ \text{and}\\ \Gamma \models^\star \alpha \text{ iff } \Gamma \models \alpha \wedge \Delta\alpha \ \Box\end{gathered}$$

Theorem 5.2 *If $\alpha \to \beta$ is an abbreviation for $\neg(\alpha \wedge \neg\beta)$, the following transformations Φ_1, Φ_2 and Φ_3 from PPC into PPC*, and Ψ_1, Ψ_2 and Ψ_3 from PPC* into PPC, are theorem preserving.*

$$\Phi_1(\Gamma \vdash \alpha) := (\Gamma \vdash^\star \Delta\alpha \to \alpha)$$

$$\Phi_1\left(\begin{array}{ccc}\Gamma_1 & \vdash & \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n\\ \hline \Gamma & \vdash & \beta\end{array}\right) := \left(\begin{array}{ccc}\Gamma_1 & \vdash^\star & \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash^\star & \alpha_n\\ \hline \Gamma & \vdash^\star & \Delta\beta \to \beta\end{array}\right), \forall n \in \mathbf{N}_0 = \{1,2,3,\ldots\}$$

$$\Phi_2(\Gamma \vdash \alpha) := (\Gamma \vdash^\star \Delta\alpha \to \alpha)$$

$$\Phi_2\left(\begin{array}{ccc}\Gamma_1 & \vdash & \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n\\ \hline \Gamma & \vdash & \beta\end{array}\right) := \left(\begin{array}{ccc}\Gamma_1 & \vdash^\star & \Delta\alpha_1 \to \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash^\star & \Delta\alpha_n \to \alpha_n\\ \hline \Gamma & \vdash^\star & \Delta\beta \to \beta\end{array}\right), \forall n \in \mathbf{N}_0$$

$$\Phi_3(\Gamma \vdash \alpha \wedge \Delta\alpha) := (\Gamma \vdash^\star \alpha)$$

$$\Phi_3\left(\begin{array}{ccc}\Gamma_1 & \vdash & \alpha_1 \wedge \Delta\alpha_1\\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n \wedge \Delta\alpha_n\\ \hline \Gamma & \vdash & \beta \wedge \Delta\beta\end{array}\right) := \left(\begin{array}{ccc}\Gamma_1 & \vdash^\star & \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash^\star & \alpha_n\\ \hline \Gamma & \vdash^\star & \beta\end{array}\right), \forall n \in \mathbf{N}_0$$

$$\Psi_1(\Gamma \vdash^\star \alpha) := (\Gamma \vdash \alpha \wedge \Delta\alpha)$$

$$\Psi_1\left(\begin{array}{ccc}\Gamma_1 & \vdash^\star & \alpha_1\\ & \vdots & \\ \Gamma_n & \vdash^\star & \alpha_n\\ \hline \Gamma & \vdash^\star & \beta\end{array}\right) := \left(\begin{array}{ccc}\Gamma_1 & \vdash & \alpha_1 \wedge \Delta\alpha_1\\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n \wedge \Delta\alpha_n\\ \hline \Gamma & \vdash & \beta \wedge \Delta\beta\end{array}\right), \forall n \in \mathbf{N}_0$$

$$\Psi_2(\Gamma \vdash^\star \alpha) := (\Gamma \vdash \alpha)$$

$$\Psi_2\left(\begin{array}{lll}\Gamma_1 & \vdash^\star & \Delta\alpha_1 \rightarrow \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash^\star & \Delta\alpha_n \rightarrow \alpha_n \\ \hline \Gamma & \vdash^\star & \beta\end{array}\right) := \left(\begin{array}{lll}\Gamma_1 & \vdash & \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n \\ \hline \Gamma & \vdash & \beta\end{array}\right), \forall n \in \mathbf{N}_0$$

$$\Psi_3(\Gamma \vdash^\star \Delta\alpha \rightarrow \alpha) := (\Gamma \vdash \alpha)$$

$$\Psi_3\left(\begin{array}{lll}\Gamma_1 & \vdash^\star & \Delta\alpha_1 \rightarrow \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash^\star & \Delta\alpha_n \rightarrow \alpha_n \\ \hline \Gamma & \vdash^\star & \Delta\beta \rightarrow \beta\end{array}\right) := \left(\begin{array}{lll}\Gamma_1 & \vdash & \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n \\ \hline \Gamma & \vdash & \beta\end{array}\right), \forall n \in \mathbf{N}_0$$

proof.

Since both PPC and PPC* are adequate, one has for every α and Γ the following:

$$(\Gamma \vdash^\star \alpha) \Longrightarrow (\Gamma \models^\star \alpha) \Longrightarrow (\Gamma \models \alpha) \Longrightarrow (\Gamma \vdash \alpha) \qquad (1)$$

$$(\Gamma \vdash \alpha) \Longleftrightarrow (\Gamma \models \alpha) \Longleftrightarrow (\Gamma \models^\star \Delta\alpha \rightarrow \alpha) \Longleftrightarrow (\Gamma \vdash^\star \Delta\alpha \rightarrow \alpha) \qquad (2)$$

$$(\Gamma \vdash^\star \alpha) \Longleftrightarrow (\Gamma \models^\star \alpha) \Longleftrightarrow (\Gamma \models \alpha \wedge \Delta\alpha) \Longleftrightarrow (\Gamma \vdash \alpha \wedge \Delta\alpha) \qquad (3)$$

It follows directly from (2) that the first defining equation of Φ_1 is theorem preserving. Now consider the second defining equation of Φ_1 and assume that

$$\begin{array}{lll}\Gamma_1 & \vdash & \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash & \alpha_n \\ \hline \Gamma & \vdash & \beta\end{array}$$

is a theorem in PPC. Assume also $\Gamma_1 \vdash^\star \alpha_1, \ldots, \Gamma_n \vdash^\star \alpha_n$. Then by (1) one has $\Gamma_1 \vdash \alpha_1, \ldots, \Gamma_n \vdash \alpha_n$. Hence from the assumed theorem, one gets $\Gamma \vdash \beta$ which leads to $\Gamma \vdash^\star \Delta\beta \rightarrow \beta$ by (2). Hence

$$\begin{array}{lll}\Gamma_1 & \vdash^\star & \alpha_1 \\ & \vdots & \\ \Gamma_n & \vdash^\star & \alpha_n \\ \hline \Gamma & \vdash^\star & \Delta\beta \rightarrow \beta\end{array}$$

is a theorem in PPC*. One can similarly prove that the other transformations are theorem preserving. □

The transformations Φ_1, Φ_2 and Ψ_1 are totally defined in the sense that they can be applied on any theorem of PPC and PPC* respectively, while Φ_3, Ψ_2 and Ψ_3 are partially defined in the sense that they can be applied only on theorems of the considered shape. In addition, Φ_3 is the inverse of Ψ_1 and Φ_2 is the inverse of Ψ_3.

The previous transformations can be used to deduce theorems in PPC* from known theorems in PPC and vice versa. In addition, they can be used to prove some theorem in PPC* by proving another theorem in PPC, maybe with the help of KARNAK, and conversely maybe with the help of KARNAK*.

Example: As an application of these transformations, KARNAK succeeded in [6, 9] to prove the PPC theorem (SeAt):

THEOREM NAME:

Self-assertion Rule (SeAt)

SYMBOLS:

α denotes a formula

Γ denotes a list

ASSUMPTIONS:

$\Gamma, \neg\alpha \ \vdash \ \alpha$

THEOREM:

$\Gamma \ \vdash \ \alpha$

PROOF:

1. $\Gamma, \neg\alpha \ \vdash \ \alpha$ assumption
2. $\alpha \ \vdash \ \alpha$ (A)
3. $\neg\Delta\alpha \ \vdash \ \neg\Delta\alpha$ (A)
4. $\neg\Delta\alpha \ \vdash \ \alpha$ (D_1) on 3
5. $\Gamma \ \vdash \ \alpha$ (R$'$) on 1,2,4 □

But since the rule (D_1) was used in step 4, this proof is not valid in PPC*. The rule (SeAt) does not hold in PPC*, but the following modified version holds:

$$(\text{SeAt}') \quad \frac{\begin{array}{lll}\Gamma_1, \neg\alpha & \vdash^\star & \alpha \\ \Gamma_2 & \vdash^\star & \Delta\alpha\end{array}}{\begin{array}{lll}\Gamma_1, \Gamma_2 & \vdash^\star & \alpha\end{array}}$$

If one applies Φ_1 on (SeAt), one gets

$$(\Phi_1(\text{SeAt})) \quad \frac{\begin{array}{lll}\Gamma, \neg\alpha & \vdash^\star & \alpha\end{array}}{\begin{array}{lll}\Gamma & \vdash^\star & \Delta\alpha \to \alpha\end{array}}$$

Since the “Modus Ponens” rule holds in PPC*, one gets (SeAt$'$) from Φ_1(SeAt) directly, and there is no need to use KARNAK* to prove (SeAt$'$) as we did in APPENDIX D of this paper.

6 Concluding Remarks

- In [1, 4] it was shown that LPF (and hence PPC*) is a logic suitable for covering partial functions in formal specifications of programs. Proofs play a vital part in the application of formal methods in software development. In VDM [10] for example, proofs establish properties of specification and realization. If proofs themselves are as error-prone as programs, the formal method will only add to an already difficult task a similarly difficult one and can not provide a solution for the so-called "software crisis". Automation of logic is a prerequisite of any system that may contribute to the solution of this problem.

- KARNAK* may be the core of a "formal software development support system" like Mural [11] to perform the routine proofs needed for verification. PPC* is equivalent to LPF (provided that the symbol "*uu*" and its rules are excluded) which is the logical basis of VDM and Mural. The Theorem Proving Assistant TPA of Mural is interactive and has a sophisticated user interface. It is mainly based on pattern matching and provides at each step a list of all the applicable rules and the user has to choose between them. Then, it provides a sophisticated interaction and a little automation. On the other hand, KARNAK* is fully automatic, but can also be used in interactive ways as shown in [6, 9] and in some of the proofs introduced in Appendix C. Then, it provides more automation and less interaction than the TPA of Mural. With respect to the implementation, KARNAK* is implemented in PDC prolog while Mural is implemented in SmallTalk. Prolog itself is a programmable theorem prover for Horn clauses based on resolution, and it is an excellent environment to develop an ATP system like KARNAK*. The choice of SmallTalk in the Mural project was mainly due to its advantages with respect to the design of user interfaces. KARNAK* is not generic like the TPA of Mural, but since KARNAK can reason about logical syntax at a fairly low level, it may be modified for other calculi (as has been done in this paper for PPC*).

- KARNAK* has been used in proving the completeness of PPC* which is a significant property of the calculus itself. In addition, it has been used in an indirect way to discover the dependency of two PPC* rules. This shows that the program is practically useful and may be assistant for mathematicians and logicians to study properties of partially defined functions and predicates.

- The transformations which are introduced in section 5 connect PPC and KARNAK on one hand and PPC* and KARNAK* on the other hand.

Appendix A

The Deduction Rules of PPC

Classical Rules

(A) $\dfrac{}{\alpha \ \vdash \ \alpha}$; (C$_0$) $\dfrac{\begin{array}{lcl}\Gamma_1 & \vdash & \alpha\\ \Gamma_2 & \vdash & \beta\end{array}}{\Gamma_1, \Gamma_2 \ \vdash \ \alpha \wedge \beta}$; (C$_i$) $\dfrac{\Gamma \ \vdash \ \alpha_1 \wedge \alpha_2}{\Gamma \ \vdash \ \alpha_i}$; $i = 1, 2$

(G) $\dfrac{\Gamma \ \vdash \ \forall x\alpha}{\Gamma \ \vdash \ \alpha}$; (G$_x$) $\dfrac{\Gamma \ \vdash \ \alpha}{\Gamma \ \vdash \ \forall x\alpha}$ if x is not free in Γ ;

(E) $\dfrac{}{\vdash \ t = t}$; (E$_x^t$) $\dfrac{\Gamma \ \vdash \ \alpha}{\Gamma, x = t \ \vdash \ \beta}$ if **subst**$xt\alpha\beta(*)$;

Modified Classical Rule

Removal rule

(R$'$) $\dfrac{\begin{array}{lcl}\Gamma_1, \alpha & \vdash & \beta\\ \Gamma_2, \neg\alpha & \vdash & \beta\\ \Gamma_3, \neg\Delta\alpha & \vdash & \beta\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3 \ \vdash \ \beta}$;

Contradiction rule

(X$'$) $\dfrac{\begin{array}{lcl}\Gamma_1 & \vdash & \alpha\\ \Gamma_2 & \vdash & \neg\alpha\\ \Gamma_3 & \vdash & \Delta\alpha\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3 \ \vdash \ \beta}$;

Substitution rule

(S$'^t_x$) $\dfrac{\begin{array}{lcl}\Gamma_1 & \vdash & \alpha\\ \Gamma_2 & \vdash & \Delta(t = t)\end{array}}{\Gamma_2, \Gamma_3 \ \vdash \ \beta}$ if **subst**$xt\Gamma_1\Gamma_3$ and **subst**$xt\alpha\beta$;

Rules concerning Δ

(Ad$_1$) $\dfrac{}{\alpha \ \vdash \ \Delta\alpha}$; (Ad$_2$) $\dfrac{}{\neg\alpha \ \vdash \ \Delta\alpha}$; (D$_0$) $\dfrac{}{\vdash \ \Delta\Delta\alpha}$;

(D$_1$) $\dfrac{\Gamma \ \vdash \ \neg\Delta\alpha}{\Gamma \ \vdash \ \alpha}$; (D$_2$) $\dfrac{\Gamma \ \vdash \ \neg\Delta\alpha}{\Gamma \ \vdash \ \neg\alpha}$; (Dn$_1$) $\dfrac{\Gamma \ \vdash \ \Delta\alpha}{\Gamma \ \vdash \ \Delta\neg\alpha}$;

(Cd$_0$) $\dfrac{\begin{array}{lcl}\Gamma_1 & \vdash & \Delta\alpha_1\\ \Gamma_2 & \vdash & \Delta\alpha_2\end{array}}{\Gamma_1, \Gamma_2 \ \vdash \ \Delta(\alpha_1 \wedge \alpha_2)}$; (Cn$_i$) $\dfrac{\begin{array}{lcl}\Gamma_1 & \vdash & \neg\alpha_i\\ \Gamma_2 & \vdash & \Delta\alpha_i\end{array}}{\Gamma_1, \Gamma_2 \ \vdash \ \Delta(\alpha_1 \wedge \alpha_2)}$; $i = 1, 2$

(Cd$_i$) $\dfrac{}{\alpha_1 \wedge \alpha_2 \ \vdash \ \Delta\alpha_i}$; $i = 1, 2$ (Gd$_1$) $\dfrac{}{\forall x\alpha \ \vdash \ \Delta\alpha}$;

(Gd$_2$) $\dfrac{\Gamma \ \vdash \ \forall x\Delta\alpha}{\Gamma \ \vdash \ \Delta\forall x\alpha}$; (Gd$_3$) $\dfrac{}{\neg\alpha \ \vdash \ \Delta\forall x\alpha}$;

(Ed$_1$) $\dfrac{}{\vdash \ \Delta(x = x)}$; (Ed$_2$) $\dfrac{}{\vdash \ \Delta(c = c)}$;

(De$_1$) $\dfrac{\Gamma \ \vdash \ \Delta(t_1 = t_2)}{\Gamma \ \vdash \ \Delta(t_i = t_i)}$; $i = 1, 2$ (De$_2$) $\dfrac{\begin{array}{lcl}\Gamma_1, \ \neg\Delta(t_1 = t_1) & \vdash & \alpha\\ \Gamma_2, \ \neg\Delta(t_2 = t_2) & \vdash & \alpha\end{array}}{\Gamma_1, \Gamma_2, \neg\Delta(t_1 = t_2) \ \vdash \ \alpha}$

(*) **subst**$xt\alpha\beta$ denotes that β is the result of substituting the term t for the variable x in α. The substitution calculus in PPC is like in the classical predicate logic with the new rule: **subst**$xt\Delta\alpha\Delta\beta$ if **subst**$xt\alpha\beta$.

Appendix B
The Deduction Rules of LPF

In this appendix, the deduction rules of both N-LPF and S-LPF are presented as in [4].

The Deduction Rules of N-LPF:

$$(\textit{def-c})\quad \frac{}{c = c} \qquad\qquad (\textit{def-v})\quad \frac{}{y = y}$$

$$(\neg\,\neg\text{-}I)\quad \frac{\alpha}{\neg\neg\alpha} \qquad\qquad (\neg\,\neg\text{-}E)\quad \frac{\neg\neg\alpha}{\alpha}$$

$$(\Delta\text{-}I\text{-}1)\quad \frac{\alpha}{\Delta\alpha} \qquad\qquad (\Delta\text{-}I\text{-}2)\quad \frac{\neg\alpha}{\Delta\alpha}$$

$$(\Delta\text{-}E)\quad \frac{\Delta\alpha; \alpha \vdash \beta; \neg\alpha \vdash \beta}{\beta}$$

$$(\neg\Delta\text{-}I)\quad \frac{\Delta\alpha \vdash \beta; \Delta\alpha \vdash \neg\beta}{\neg\Delta\alpha} \qquad\qquad (\neg\Delta\text{-}E)\quad \frac{\neg\Delta\alpha \vdash \beta; \neg\Delta\alpha \vdash \neg\beta}{\Delta\alpha}$$

$$(\vee\text{-}I\text{-}L)\quad \frac{\alpha}{\alpha \vee \beta} \qquad\qquad (\vee\text{-}I\text{-}R)\quad \frac{\beta}{\alpha \vee \beta}$$

$$(\vee\text{-}E)\quad \frac{\alpha_1 \vee \alpha_2; \alpha_1 \vdash \beta; \alpha_2 \vdash \beta}{\beta}$$

$$(\neg\,\vee\text{-}I)\quad \frac{\neg\alpha; \neg\beta}{\neg(\alpha \vee \beta)} \qquad\qquad (\neg\,\vee\text{-}E\text{-}L)\quad \frac{\neg(\alpha \vee \beta)}{\neg\alpha}$$

$$(\neg\,\vee\text{-}E\text{-}R)\quad \frac{\neg(\alpha \vee \beta)}{\neg\beta}$$

$$(\exists\text{-}I)\quad \frac{\alpha(x/t); t = t}{\exists x\alpha} \qquad\qquad (\exists\text{-}E)\quad \frac{\alpha \vdash \beta; \exists x\alpha}{\beta}(*)$$

$$(\neg\,\exists\text{-}I)\quad \frac{\neg\alpha}{\neg\exists x\alpha}(**) \qquad\qquad (\neg\,\exists\text{-}E)\quad \frac{\neg\exists x\alpha; t = t}{\neg\alpha(x/t)}$$

(*uu-E*) $\dfrac{uu}{\alpha}$ (¬*uu-E*) $\dfrac{\neg uu}{\alpha}$

(*contr*) $\dfrac{\alpha; \neg\alpha}{\beta}$ (¬ *E-E*) $\dfrac{\neg(t = t)}{\alpha}$

(=-*reflx*) $\dfrac{t_1 = t_2}{t_1 = t_1}$

(¬ =-*reflx-L*) $\dfrac{\neg(t_1 = t_2)}{t_1 = t_1}$ (¬ =-*reflx-R*) $\dfrac{\neg(t_1 = t_2)}{t_2 = t_2}$

(=-*subs1*) $\dfrac{t_1 = t_2; \alpha(y/t_1)}{\alpha(y/t_2)}$ (=-*2-val*) $\dfrac{t_1 = t_1; t_2 = t_2}{(t_1 = t_2) \vee \neg(t_1 = t_2)}$

(=-*subs2*) $\dfrac{t_1 = t_2; t(y/t_1) = t(y/t_1)}{t(y/t_1) = t(y/t_2)}$

(*) x is not free in β and is not free in any assumption except α on which β depends.
(**) x is not free in any assumption on which $\neg\alpha$ depends.

In the previous rules, the notation $\alpha \vdash \beta$ corresponds to the standard notation

$$\begin{matrix}[\alpha]\\ \vdots \\ \beta\end{matrix}$$

The semicolon ";" is used to separate between assumptions in each rule, while the comma "," is used to separate between formulas in the antecedent of each sequent. Hence, in (Δ-*E*) for example the assumptions are "$\Delta\alpha$", "$\alpha \vdash \beta$", "$\neg\alpha \vdash \beta$", and not "$\Delta\alpha, \alpha \vdash \beta$", "$\neg\alpha \vdash \beta$".

The variable y in the rules (=-*subs1*) and (=-*subs2*) is arbitrary and plays the role of a place holder. Besides, $\alpha(y/t)$ denotes the result of replacing the variable y by the term t in α.

The Deduction Rules of S-LPF:

(1) $\alpha \longrightarrow \alpha$ (2) $\alpha, \neg\alpha \longrightarrow$ (3) $uu \longrightarrow$

(4) $\neg uu \longrightarrow$ (5) $\longrightarrow c = c$ (6) $\longrightarrow y = y$

(7) $\neg(t = t) \longrightarrow$ (8) $\dfrac{\begin{matrix}\neg\alpha, \Gamma \longrightarrow \Sigma\\ \alpha, \Gamma \longrightarrow \Sigma\end{matrix}}{\Delta\alpha, \Gamma \longrightarrow \Sigma}$ (9) $\dfrac{\begin{matrix}\neg\alpha, \Gamma \longrightarrow \Sigma\\ \alpha, \Gamma \longrightarrow \Sigma\end{matrix}}{\Gamma \longrightarrow \Sigma, \neg\Delta\alpha}$

(10.a) $\dfrac{\Gamma \longrightarrow \Sigma, \alpha}{\Gamma \longrightarrow \Sigma, \Delta\alpha}$ (10.b) $\dfrac{\Gamma \longrightarrow \Sigma, \neg\alpha}{\Gamma \longrightarrow \Sigma, \Delta\alpha}$

(11.a) $\dfrac{\Gamma \longrightarrow \Sigma, \alpha}{\neg\Delta\alpha, \Gamma \longrightarrow \Sigma}$ (11.b) $\dfrac{\Gamma \longrightarrow \Sigma, \neg\alpha}{\neg\Delta\alpha, \Gamma \longrightarrow \Sigma}$

$$(12)\quad \frac{\alpha, \Gamma \longrightarrow \Sigma}{\neg\neg\alpha, \Gamma \longrightarrow \Sigma} \qquad (13)\quad \frac{\Gamma \longrightarrow \Sigma, \alpha}{\Gamma \longrightarrow \Sigma, \neg\neg\alpha} \qquad (14)\quad \frac{\begin{array}{l}\alpha, \Gamma \longrightarrow \Sigma \\ \beta, \Gamma \longrightarrow \Sigma\end{array}}{\alpha \vee \beta, \Gamma \longrightarrow \Sigma}$$

$$(15.a)\quad \frac{\Gamma \longrightarrow \Sigma, \alpha}{\Gamma \longrightarrow \Sigma, \alpha \vee \beta} \qquad (15.b)\quad \frac{\Gamma \longrightarrow \Sigma, \beta}{\Gamma \longrightarrow \Sigma, \alpha \vee \beta}$$

$$(16.a)\quad \frac{\neg\alpha, \Gamma \longrightarrow \Sigma}{\neg(\alpha \vee \beta), \Gamma \longrightarrow \Sigma} \qquad (16.b)\quad \frac{\neg\beta, \Gamma \longrightarrow \Sigma}{\neg(\alpha \vee \beta), \Gamma \longrightarrow \Sigma}$$

$$(17)\quad \frac{\begin{array}{l}\Gamma \longrightarrow \Sigma, \neg\alpha \\ \Gamma \longrightarrow \Sigma, \neg\beta\end{array}}{\Gamma \longrightarrow \Sigma, \neg(\alpha \vee \beta)} \qquad (18)\quad \frac{\alpha, \Gamma \longrightarrow \Sigma}{\exists x\alpha, \Gamma \longrightarrow \Sigma}(***) \qquad (19)\quad \frac{\begin{array}{l}\Gamma \longrightarrow \Sigma, t = t \\ \Gamma \longrightarrow \Sigma, \alpha(x/t)\end{array}}{\Gamma \longrightarrow \Sigma, \exists x\alpha}$$

$$(20)\quad \frac{\begin{array}{l}\Gamma \longrightarrow \Sigma, t = t \\ \neg\alpha(x/t), \Gamma \longrightarrow \Sigma\end{array}}{\neg\exists x\alpha, \Gamma \longrightarrow \Sigma} \qquad (21)\quad \frac{\Gamma \longrightarrow \Sigma, \neg\alpha}{\Gamma \longrightarrow \Sigma, \neg\exists x\alpha}(***)$$

$$(22)\quad \frac{\Gamma \longrightarrow \Sigma}{\alpha, \Gamma \longrightarrow \Sigma}$$

$$(23)\quad \frac{\Gamma \longrightarrow \Sigma}{\Gamma \longrightarrow \Sigma, \alpha}$$

$$(24)\quad t_1 = t_2, t(y/t_1) = t(y/t_1) \longrightarrow t(y/t_1) = t(y/t_2)$$

$$(25)\quad t_1 = t_2, \alpha(y/t_1) \longrightarrow \alpha(y/t_2)$$

$$(26)\quad t_1 = t_2 \longrightarrow t_1 = t_1$$

$$(27)\quad \neg(t_1 = t_2) \longrightarrow t_1 = t_1$$

$$(28)\quad \neg(t_1 = t_2) \longrightarrow t_2 = t_2$$

$$(29)\quad t_1 = t_1, t_2 = t_2 \longrightarrow t_1 = t_2, \neg(t_1 = t_2)$$

(***) x is not free in $\Gamma \cup \Sigma$.

The cut rule was first introduced with the S-LPF deduction rules and then was proven to be dependent. Therefore, it is not written with the previous rules (see chapter 5 in [4]).

Appendix C

Some of the KARNAK* Proofs for the N-LPF$'$ Deduction Rules

In the following, some of the KARNAK* proofs for the N-LPF$'$ deduction rules are introduced. The existential quantifier $\exists x$ and the disjunction $\alpha \vee \beta$ in N-LPF$'$ are replaced in PPC* by $\neg\forall x\neg$ and $\neg(\neg\alpha \wedge \neg\beta)$ respectively.

- $\longrightarrow$

```
include"karnak#.pro"
clauses
deduce(sequent([n(st(alpha)),n(st(beta))],
                n(n(a(n(st(alpha)),n(st(beta))))))).
```

$\Leftarrow$

THEOREM NAME:

$\neg\ \vee$-I

SYMBOLS:

α,β denote formulas

THEOREM:

$\neg\alpha,\ \neg\beta\ \ \vdash^\star\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$

PROOF:

1. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \Delta\neg\alpha$ (Cd$_1$)
2. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \Delta\neg\beta$ (Cd$_2$)
3. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \Delta\beta$ (Dn$_2$) on 2
4. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \Delta\alpha$ (Dn$_2$) on 1
5. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \neg\alpha\wedge\neg\beta$ (A)
6. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \neg\alpha$ (C$_1$) on 5
7. $\neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \neg\beta$ (C$_2$) on 5
8. $\neg\alpha\ \ \vdash^\star\ \ \neg\alpha$ (A)
9. $\neg\beta\ \ \vdash^\star\ \ \neg\beta$ (A)
10. $\neg\alpha,\ \neg\beta\ \ \vdash^\star\ \ \neg\alpha\wedge\neg\beta$ (C$_0$) on 8,9
11. $\neg\alpha,\ \neg\beta\ \ \vdash^\star\ \ \Delta(\neg\alpha\wedge\neg\beta)$ (Δ) on 10
12. $\neg\alpha,\ \neg\beta\ \ \vdash^\star\ \ \Delta\neg(\neg\alpha\wedge\neg\beta)$ (Dn$_1$) on 11
13. $\neg\Delta\alpha\ \ \vdash^\star\ \ \neg\Delta\alpha$ (A)
14. $\neg\Delta\alpha,\ \neg\alpha\wedge\neg\beta\ \ \vdash^\star\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 4,13
15. $\alpha\ \ \vdash^\star\ \ \alpha$ (A)
16. $\neg\alpha\wedge\neg\beta,\ \alpha\ \ \vdash^\star\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 6,15
17. $\neg\Delta\beta\ \ \vdash^\star\ \ \neg\Delta\beta$ (A)

18. $\neg\Delta\beta,\ \neg\alpha\wedge\neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 3,17
19. $\beta\ \ \vdash^{\star}\ \ \beta$ (A)
20. $\neg\alpha\wedge\neg\beta,\ \beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 7,19
21. $\neg\Delta\neg(\neg\alpha\wedge\neg\beta)\ \ \vdash^{\star}\ \ \neg\Delta\neg(\neg\alpha\wedge\neg\beta)$ (A)
22. $\neg\Delta\neg(\neg\alpha\wedge\neg\beta),\ \neg\alpha,\ \neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 12,21
23. $\neg\Delta(\neg\alpha\wedge\neg\beta)\ \ \vdash^{\star}\ \ \neg\Delta(\neg\alpha\wedge\neg\beta)$ (A)
24. $\neg\Delta(\neg\alpha\wedge\neg\beta),\ \neg\alpha,\ \neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 11,23
25. $\neg(\neg\alpha\wedge\neg\beta)\ \ \vdash^{\star}\ \ \neg(\neg\alpha\wedge\neg\beta)$ (A)
26. $\neg(\neg\alpha\wedge\neg\beta),\ \neg\alpha,\ \neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (X) on 10,25
27. $\neg\alpha\wedge\neg\beta,\ \neg(\neg\alpha\wedge\neg\beta),\ \neg\alpha\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (R′) on 18,20,26
28. $\neg\alpha\wedge\neg\beta,\ \neg(\neg\alpha\wedge\neg\beta)\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (R′) on 14,16,27
29. $\neg\neg(\neg\alpha\wedge\neg\beta)\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (A)
30. $\neg\alpha,\ \neg\beta,\ \neg\alpha\wedge\neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (R′) on 22,28,29
31. $\neg\alpha,\ \neg\beta\ \ \vdash^{\star}\ \ \neg\neg(\neg\alpha\wedge\neg\beta)$ (R′) on 24,26,30 □

$\longleftarrow$

```
PT =     247.05   sec;          WT =       5.60   sec;
DL =      91      steps;        PL =      31      steps;
CO =      34.07   %;            DE =       1.13;
DS =       0.13   steps/sec;    ST =       7.97   sec.
```

Although the previous proof was carried out on the propositional level (does not contain quantifiers or substitutions), it looks difficult and tedious. However, the proving time is approximately 4 minutes. If one tries to discover and write this proof manually, it is really difficult to achieve that in this time. This illustrates the efficiency of the program. Moreover, in some cases the proving time is only a fraction of a second.

• $\longrightarrow$

```
include"karnak#.pro"
clauses
formula(e(v(y),t(t_2))).
assume(notin(v(y),t(t_2))).
assume(subst(v(y),t(t_1),st(alpha),st(alpha1))).
assume(subst(v(y),t(t_2),st(alpha),st(alpha2))).
assume(sequent([st(alpha)],st(alpha))).
deduce(sequent([e(t(t_1),t(t_2)),st(alpha1)],st(alpha2))).
```

$\Leftarrow$

THEOREM NAME:

=-subs1

SYMBOLS:

y denotes a variable

t_2,t_1 denote terms

α,α_1,α_2 denote formulas

ASSUMPTIONS:

y is not in t_2

subst y t_1 α α_1

subst y t_2 α α_2

$\alpha \ \vdash^\star \ \alpha$

THEOREM:

$t_1{=}t_2,\ \alpha_1 \ \vdash^\star \ \alpha_2$

PROOF:

1. y is not in t_2 assumption
2. subst y t_1 α α_1 assumption
3. subst y t_2 α α_2 assumption
4. $\alpha \ \vdash^\star \ \alpha$ assumption
5. y is not free in α_2 by 1,3
6. $\alpha,\ y{=}t_2 \ \vdash^\star \ \alpha_2$ $(\mathrm{E}_y^{t_2})$ on 3,4
7. $t_1{=}t_2 \ \vdash^\star \ \Delta(t_1{=}t_2)$ (Ad_1)
8. $t_1{=}t_2 \ \vdash^\star \ \Delta(t_1{=}t_1)$ (De_1) on 7
9. $t_1{=}t_2,\ \alpha_1 \ \vdash^\star \ \alpha_2$ $(\mathrm{S}'^{t_1}_y)$ on 1,2,5,6,8 □

$\longleftarrow$

```
PT =      1.53   sec;          WT =      0.27   sec;
DL =     28      steps;        PL =      9      steps;
CO =     32.14   %;            DE =      1.22;
DS =      5.88   steps/sec;    ST =      0.17   sec.
```

One may notice that the assumption $\alpha \vdash^\star \alpha$ is redundant. However, it has been added because we already imagine the sketch of the proof, where α_2 can be obtained in the succedent by applying $(\mathrm{E}_y^{t_2})$ and then α_1 can be obtained in the antecedent by applying $(\mathrm{S}'^{t_1}_y)$. Therefore, the assumption "y is not in t_2" was also added. If the assumption $\alpha \vdash^\star \alpha$ was not assumed, the program may take some time to generate this step by applying (A). Besides, the addition of

```
formula(e(v(y),t(t_2))).
```

in the KTS theorem shows how "principle 8" helps to control the application of (E_x^t) by introducing certain pairs of x and t [6, 9]. One may also notice that this database clause appears implicitly in step 6 of the written proof.

This example shows how the user designs a sketch of the proof by himself and then uses the program to perform the actual proof and fill in the details.

- $\longrightarrow$

```
include"karnak#.pro"
clauses
assume(sequent([l("Gamma1")],e(t(t_1),t(t_1)))).
assume(sequent([l("Gamma2")],e(t(t_2),t(t_2)))).
deduce(sequent([l("Gamma1"),l("Gamma2")],
               d(a(n(e(t(t_1),t(t_2))),n(n(e(t(t_1),t(t_2))))))).
```

$\Leftarrow$

THEOREM NAME:

Lemma for =-*2-val*

SYMBOLS:

t_1,t_2 denote terms

Γ_1,Γ_2 denote lists

ASSUMPTIONS:

$\Gamma_1 \vdash^\star t_1{=}t_1$

$\Gamma_2 \vdash^\star t_2{=}t_2$

THEOREM:

$\Gamma_1, \Gamma_2 \vdash^\star \Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$

PROOF:

1. $\Gamma_1 \vdash^\star t_1{=}t_1$ assumption
2. $\Gamma_2 \vdash^\star t_2{=}t_2$ assumption
3. $\Gamma_2 \vdash^\star \Delta(t_2{=}t_2)$ (Δ) on 2
4. $\Gamma_1 \vdash^\star \Delta(t_1{=}t_1)$ (Δ) on 1
5. $\neg\Delta(t_1{=}t_1) \vdash^\star \neg\Delta(t_1{=}t_1)$ (A)
6. $\neg\Delta(t_1{=}t_1), \Gamma_1 \vdash^\star \neg\neg(t_1{=}t_2)$ (X) on 4,5
7. $\neg\Delta(t_1{=}t_1), \Gamma_1 \vdash^\star \Delta\neg\neg(t_1{=}t_2)$ (Δ) on 6
8. $\neg\Delta(t_1{=}t_1), \Gamma_1 \vdash^\star \Delta\neg(t_1{=}t_2)$ (Dn_2) on 7
9. $\neg\Delta(t_1{=}t_1), \Gamma_1 \vdash^\star \Delta(t_1{=}t_2)$ (Dn_2) on 8
10. $\neg\Delta(t_2{=}t_2) \vdash^\star \neg\Delta(t_2{=}t_2)$ (A)
11. $\neg\Delta(t_2{=}t_2), \Gamma_2 \vdash^\star \neg\neg(t_1{=}t_2)$ (X) on 3,10
12. $\neg\Delta(t_2{=}t_2), \Gamma_2 \vdash^\star \Delta\neg\neg(t_1{=}t_2)$ (Δ) on 11
13. $\neg\Delta(t_2{=}t_2), \Gamma_2 \vdash^\star \Delta\neg(t_1{=}t_2)$ (Dn_2) on 12
14. $\neg\Delta(t_2{=}t_2), \Gamma_2 \vdash^\star \Delta(t_1{=}t_2)$ (Dn_2) on 13
15. $\Gamma_1, \Gamma_2, \neg\Delta(t_1{=}t_2) \vdash^\star \Delta(t_1{=}t_2)$ (De_2) on 9,14
16. $\Gamma_1, \Gamma_2, \neg\Delta(t_1{=}t_2) \vdash^\star \Delta\neg(t_1{=}t_2)$ (Dn_1) on 15
17. $\Gamma_1, \Gamma_2, \neg\Delta(t_1{=}t_2) \vdash^\star \Delta\neg\neg(t_1{=}t_2)$ (Dn_1) on 16

18. $\Gamma_1, \Gamma_2, \neg\Delta(t_1{=}t_2) \;\vdash^\star\; \Delta\neg\neg\neg(t_1{=}t_2)$ (Dn$_1$) on 17
19. $\neg(t_1{=}t_2) \;\vdash^\star\; \Delta\neg(t_1{=}t_2)$ (Ad$_1$)
20. $\neg(t_1{=}t_2) \;\vdash^\star\; \Delta\neg\neg(t_1{=}t_2)$ (Dn$_1$) on 19
21. $\neg(t_1{=}t_2) \;\vdash^\star\; \Delta\neg\neg\neg(t_1{=}t_2)$ (Dn$_1$) on 20
22. $t_1{=}t_2 \;\vdash^\star\; \Delta(t_1{=}t_2)$ (Ad$_1$)
23. $t_1{=}t_2 \;\vdash^\star\; \Delta\neg(t_1{=}t_2)$ (Dn$_1$) on 22
24. $t_1{=}t_2 \;\vdash^\star\; \Delta\neg\neg(t_1{=}t_2)$ (Dn$_1$) on 23
25. $t_1{=}t_2 \;\vdash^\star\; \Delta\neg\neg\neg(t_1{=}t_2)$ (Dn$_1$) on 24
26. $\Gamma_1, \Gamma_2 \;\vdash^\star\; \Delta\neg\neg\neg(t_1{=}t_2)$ (R$'$) on 18,21,25
27. $\Gamma_1, \Gamma_2 \;\vdash^\star\; \Delta\neg\neg(t_1{=}t_2)$ (Dn$_2$) on 26
28. $\Gamma_1, \Gamma_2 \;\vdash^\star\; \Delta\neg(t_1{=}t_2)$ (Dn$_2$) on 27
29. $\Gamma_1, \Gamma_2 \;\vdash^\star\; \Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (Cd$_0$) on 27,28 □

$\longleftarrow$

```
PT =     164.50   sec;          WT =      11.42   sec;
DL =     131      steps;        PL =      29      steps;
CO =      22.14   %;            DE =       1.03;
DS =       0.18   steps/sec;    ST =       5.67   sec.
```

$\longrightarrow$

```
include"karnak#.pro"
clauses
assume(sequent([l("Gamma1")],e(t(t_1),t(t_1)))).
assume(sequent([l("Gamma2")],e(t(t_2),t(t_2)))).
assume(sequent([l("Gamma1"),l("Gamma2")],
               d(a(n(e(t(t_1),t(t_2))),n(n(e(t(t_1),t(t_2)))))))).
deduce(sequent([l("Gamma1"),l("Gamma2")],
               n(a(n(e(t(t_1),t(t_2))),n(n(e(t(t_1),t(t_2)))))))).
```

$\Leftarrow$

THEOREM NAME:

The remaining part of =-*2-val*

SYMBOLS:

t_1,t_2 denote terms

Γ_1,Γ_2 denote lists

ASSUMPTIONS:

$\Gamma_1 \;\vdash^\star\; t_1{=}t_1$

$\Gamma_2 \;\vdash^\star\; t_2{=}t_2$

$\Gamma_1, \Gamma_2 \;\vdash^\star\; \Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$

THEOREM:

$\Gamma_1, \Gamma_2 \ \vdash^\star \ \neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$

PROOF:

1. $\Gamma_1, \Gamma_2 \ \vdash^\star \ \Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ assumption
2. $\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2) \ \vdash^\star \ \neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2)$ (A)
3. $\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2) \ \vdash^\star \ \neg(t_1{=}t_2)$ (C_1) on 2
4. $\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2) \ \vdash^\star \ \neg\neg(t_1{=}t_2)$ (C_2) on 2
5. $\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2) \ \vdash^\star \ \neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (X) on 3,4
6. $\neg\Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2)) \ \vdash^\star \ \neg\Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (A)
7. $\neg\Delta(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2)), \Gamma_1, \Gamma_2 \ \vdash^\star \ \neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (X) on 1,6
8. $\neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2)) \ \vdash^\star \ \neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (A)
9. $\Gamma_1, \Gamma_2 \ \vdash^\star \ \neg(\neg(t_1{=}t_2)\wedge\neg\neg(t_1{=}t_2))$ (R′) on 5,7,8 □

$\longleftarrow$

```
PT =      17.85   sec;            WT =       1.32   sec;
DL =      44      steps;          PL =       9      steps;
CO =      20.45   %;              DE =       1.11;
DS =       0.50   steps/sec;      ST =       1.98   sec.
```

In the previous proof, we used one of the interaction methods which are explained in [9], specifically proving a theorem through intermediate lemmas. The program may fail to prove some theorem directly and succeed to prove it through intermediate lemmas. This is because the non-normal proofs may be normal if one breaks them into partial proofs. In order to clarify this, suppose that a proof of some theorem contains in the kth step a conjunction $\alpha \wedge \beta$ which does not exist in the theorem. Then, the program will not be able to discover this proof due to "principle 1". However, if one gives to the program the assumptions of the theorem together with the kth step as a conjecture, then it may be able to prove this lemma because the conjunction $\alpha \wedge \beta$ exists in it, unlike the original theorem.

- $\longrightarrow$

```
include"karnak#.pro"
clauses
formula(e(v(x),t(t_2))).
assume(replace(v(y),t(t_1),t(t),t("t(y/t_1)"))).
assume(replace(v(y),t(t_2),t(t),t("t(y/t_2)"))).
assume(replace(v(y),v(x),t(t),t("t(y/x)"))).
assume(replace(v(x),t(t_2),t("t(y/x)"),t("t(y/t_2)"))).
assume(replace(v(x),t(t_1),t("t(y/x)"),t("t(y/t_1)"))).
assume(notin(v(x),t(t))).
assume(notin(v(x),t(t_1))).
assume(notin(v(x),t(t_2))).
assume(sequent([e(t("t(y/t_1)"),t("t(y/x)"))],
               e(t("t(y/t_1)"),t("t(y/x)")))).
deduce(sequent([e(t(t_1),t(t_2)),e(t("t(y/t_1)"),t("t(y/t_1)"))],
               e(t("t(y/t_1)"),t("t(y/t_2)")))).
```

$\Leftarrow$

THEOREM NAME:

=-subs2

SYMBOLS:

x,y denote variables

t_2,t_1,t,$t(y/t_1)$,$t(y/t_2)$,$t(y/x)$ denote terms

ASSUMPTIONS:

subst y t_1 t $t(y/t_1)$

subst y t_2 t $t(y/t_2)$

subst y x t $t(y/x)$

subst x t_2 $t(y/x)$ $t(y/t_2)$

subst x t_1 $t(y/x)$ $t(y/t_1)$

x is not in t

x is not in t_1

x is not in t_2

$t(y/t_1){=}t(y/x) \quad \vdash^\star \quad t(y/t_1){=}t(y/x)$

THEOREM:

$t_1{=}t_2,\ t(y/t_1){=}t(y/t_1) \quad \vdash^\star \quad t(y/t_1){=}t(y/t_2)$

PROOF:

1. subst x t_2 $t(y/x)$ $t(y/t_2)$ assumption
2. subst x t_1 $t(y/x)$ $t(y/t_1)$ assumption
3. x is not in t_1 assumption
4. x is not in t_2 assumption
5. $t(y/t_1){=}t(y/x) \quad \vdash^\star \quad t(y/t_1){=}t(y/x)$ assumption
6. x is not in $t(y/t_1)$ by 2,3
7. x is not in $t(y/t_2)$ by 1,4
8. $t(y/t_1){=}t(y/x),\ x{=}t_2 \quad \vdash^\star \quad t(y/t_1){=}t(y/t_2)$ ($\mathrm{E}_x^{t_2}$) on 1,5,6
9. $t_1{=}t_2 \quad \vdash^\star \quad \Delta(t_1{=}t_2)$ (Ad_1)
10. $t_1{=}t_2 \quad \vdash^\star \quad \Delta(t_1{=}t_1)$ (De_1) on 9
11. $t_1{=}t_2,\ t(y/t_1){=}t(y/t_1) \quad \vdash^\star \quad t(y/t_1){=}t(y/t_2)$ ($\mathrm{S'}_x^{t_1}$) on 2,4,6,7,8,10 $\square$

$\longleftarrow$

```
PT =     42.68   sec;          WT =      2.09   sec;
DL =     77      steps;        PL =     11      steps;
CO =     14.29   %;            DE =      1.36;
DS =      0.26   steps/sec;    ST =      3.88   sec.
```

In the previous proof, one may notice that the assumption

$$t(y/t_1){=}t(y/x) \quad \vdash^\star \quad t(y/t_1){=}t(y/x)$$

takes the form $\alpha \vdash^\star \alpha$. In addition, the variable x and the assumptions about it do not appear in the original theorem. The explanation of that is much similar to the case in the proof of (=-*subs1*). The assumptions about the variable x are discharged assumptions in the sense that they can be preceded by “Let”. Using discharged assumptions is another one of the interaction methods which are explained in [9]. In addition, the reader may notice that some of the assumptions are not used in the proof. This shows how the program may help to reduce the number of assumptions where the unused ones do not appear in the proof.

Appendix D

The KARNAK* Proofs for (CuRu), (Gd$_2$) and (SeAt$'$)

• $\longrightarrow$

```
include "karnak#.pro"
clauses
assume(sequent([l("Gamma1")],st(alpha))).
assume(sequent([l("Gamma2"),st(alpha)],st(beta))).
deduce(sequent([l("Gamma1"),l("Gamma2")],st(beta))).
```

$\Leftarrow$

THEOREM NAME:

Cut Rule (CuRu)

SYMBOLS:

α,β denote formulas

Γ_1,Γ_2 denote lists

ASSUMPTIONS:

$\Gamma_1 \ \vdash^\star \ \alpha$

$\Gamma_2, \alpha \ \vdash^\star \ \beta$

THEOREM:

$\Gamma_1, \Gamma_2 \ \vdash^\star \ \beta$

PROOF:

1. $\Gamma_1 \ \vdash^\star \ \alpha$ assumption
2. $\Gamma_2, \alpha \ \vdash^\star \ \beta$ assumption
3. $\Gamma_1 \ \vdash^\star \ \Delta\alpha$ (Δ) on 1
4. $\neg\Delta\alpha \ \vdash^\star \ \neg\Delta\alpha$ (A)
5. $\neg\Delta\alpha, \Gamma_1 \ \vdash^\star \ \beta$ (X) on 3,4
6. $\neg\alpha \ \vdash^\star \ \neg\alpha$ (A)
7. $\neg\alpha, \Gamma_1 \ \vdash^\star \ \beta$ (X) on 1,6
8. $\Gamma_1, \Gamma_2 \ \vdash^\star \ \beta$ (R$'$) on 2,5,7 □

$\longleftarrow$

```
PT =       0.17   sec;          WT =       0.06   sec;
DL =      17      steps;        PL =       8      steps;
CO =      47.06   %;            DE =       1.12;
DS =      47.06   steps/sec;    ST =       0.02   sec.
```

- In this proof, we use a version of KARNAK* called "`karnak#1.pro`" which does not contain (Gd_2). When we tried to prove the dependency of (Gd_2) manually, we did not succeed. This shows that KARNAK* is practically useful. In fact, it may be difficult to discover natural proofs even if they are short, although it is always easy to understand and verify them even if they are long.

$\longrightarrow$

```
include "karnak#1.pro"
clauses
deduce(sequent([for(v(x),d(st(alpha)))],d(for(v(x),st(alpha))))).
```

$\Leftarrow$

THEOREM NAME:

(Gd_2)

SYMBOLS:

x denotes a variable

α denotes a formula

THEOREM:

$\forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$

PROOF:

1. $\neg\alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (Gd_3)
2. $\forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \forall\, x\, \Delta\alpha$ (A)
3. $\forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \Delta\alpha$ (G) on 2
4. $\neg\Delta\alpha \;\; \vdash^\star \;\; \neg\Delta\alpha$ (A)
5. $\neg\Delta\alpha, \forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (X) on 3,4
6. $\neg\Delta(\forall\, x\, \alpha) \;\; \vdash^\star \;\; \neg\Delta(\forall\, x\, \alpha)$ (A)
7. $\forall\, x\, \alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (Ad_1)
8. $\neg(\forall\, x\, \alpha) \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (Ad_2)
9. $\alpha \;\; \vdash^\star \;\; \alpha$ (A)
10. $\neg\Delta(\forall\, x\, \alpha), \neg\Delta\alpha, \forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \alpha$ (X) on 5,6
11. $\neg\Delta(\forall\, x\, \alpha), \neg\alpha \;\; \vdash^\star \;\; \alpha$ (X) on 1,6
12. $\neg\Delta(\forall\, x\, \alpha), \forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \alpha$ (R$'$) on 9,10,11
13. $\neg\Delta(\forall\, x\, \alpha), \forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \forall\, x\, \alpha$ (G_x) on 12
14. $\neg\Delta(\forall\, x\, \alpha), \forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (Δ) on 13
15. $\forall\, x\, \Delta\alpha \;\; \vdash^\star \;\; \Delta(\forall\, x\, \alpha)$ (R$'$) on 7,8,14 □

$\longleftarrow$

```
PT =    2682.83   sec;          WT =      12.47   sec;
DL =     168      steps;        PL =      15      steps;
CO =       8.93   %;            DE =       1.07;
DS =       0.01   steps/sec;    ST =     178.86   sec.
```

- $\longrightarrow$

```
include "karnak#.pro"
clauses
assume(sequent([l("Gamma1"),n(st(alpha))],st(alpha))).
assume(sequent([l("Gamma2")],d(st(alpha)))).
deduce(sequent([l("Gamma1"),l("Gamma2")],st(alpha))).
```

$\Leftarrow$

THEOREM NAME:

Self-assertion Rule (SeAt′)

SYMBOLS:

α denotes a formula

Γ_1,Γ_2 denote lists

ASSUMPTIONS:

$\Gamma_1, \neg\alpha \ \ \vdash^\star \ \ \alpha$

$\Gamma_2 \ \ \vdash^\star \ \ \Delta\alpha$

THEOREM:

$\Gamma_1, \Gamma_2 \ \ \vdash^\star \ \ \alpha$

PROOF: b

1. $\Gamma_1, \neg\alpha \ \ \vdash^\star \ \ \alpha$ assumption
2. $\Gamma_2 \ \ \vdash^\star \ \ \Delta\alpha$ assumption
3. $\neg\Delta\alpha \ \ \vdash^\star \ \ \neg\Delta\alpha$ (A)
4. $\neg\Delta\alpha, \Gamma_2 \ \ \vdash^\star \ \ \alpha$ (X) on 2,3
5. $\alpha \ \ \vdash^\star \ \ \alpha$ (A)
6. $\Gamma_2, \Gamma_1 \ \ \vdash^\star \ \ \alpha$ (R′) on 1,4,5 □

$\longleftarrow$

```
PT =        0.17   sec;           WT =        0.00   sec;
DL =        9      steps;         PL =        6      steps;
CO =       66.67   %;             DE =        1.00;
DS =       35.29   steps/sec;     ST =        0.03   sec.
```

Acknowledgments: Invaluable comments were received from anonymous referees and J.H.Cheng.

References

[1] Barringer H., Cheng J.H., Jones C.B., *A Logic Covering Undefinedness in Program Proofs*, Acta Informatica 21, 251-269 (1984).

[2] Bolc L., Borowik P., *Many-Valued Logics. 1 Theoretical Foundations*, Springer Verlag (1992).

[3] Bowen K.A., *Programming with Full First-Order Logic*, MI 10, 421-440 (1982).

[4] Cheng J., *A Logic for Partial Functions*, PhD Thesis, Department of Computer Science, Manchester University (1986).

[5] Duffy D., *Principles of Automated Theorem Proving*, Wiley (1991).

[6] Elnadi T.M., Hoogewijs A., *An Automated Theorem Prover for the Partial Predicate Calculus PPC*, submitted to Journal of Automated Reasoning (1994).

[7] Hoogewijs A., *A Calculus of Partially Defined Predicates*, Mathematical Scripts, Gent University (1977).

[8] Hoogewijs A., *Partial-Predicate Logic in Computer Science*, Acta Informatica 24, 381-393 (1987).

[9] Hoogewijs A., Elnadi T.M., *KARNAK: An Automated Theorem Prover for PPC*, The CAGe Reports No. 10, Gent (1994).

[10] Jones C., *VDM Proof Obligations and their Justification*, Lecture Notes in Computer Science 252, Springer Verlag, 260-286 (1987).

[11] Jones C., Jones K., Lindsay P., Moore R., *Mural: A Formal Development Support System*, Springer Verlag (1991).

[12] Kanger S., *A Simplified Proof Method for Elementary Logic*, Computer Programming and Formal Systems, Eds. P. Braffort, D. Hirshberg. North Holland, 87-94 (1963).

[13] Malinowski G., *Many-Valued Logics*, Oxford Logic Guides 25, Oxford (1993).

[14] Ryan M., Sadler M., *Valuation Systems and Consequence Relations* in *Handbook of Logic in Computer Science Vol. 1*, Clarendon Press, Oxford, 1-78 (1992).

[15] Wang H., *Towards Mechanical Mathematics*, IBM J. Res. Dev. 4, 2-22 (1960).

Tarek Mohamed Elnadi
Mathematics Department, Faculty of Science,
Mansoura University, Egypt
ten@cage.rug.ac.be

Albert Hoogewijs
Department of Pure Mathematics and Computer Algebra,
Gent University, Belgium
bh@cage.rug.ac.be