LETTER *Special Section on Foundations of Computer Science —Frontiers of Theory of Computation and Algorithm—*

A Heuristic Proof Procedure for First-Order Logic

Keehang KWON[†a)], ***Member***

SUMMARY Inspired by the efficient proof procedures discussed in *Computability logic* [3], [5], [6], we describe a heuristic proof procedure for first-order logic. This is a variant of Gentzen sequent system [2] and has the following features: (a) it views sequents as games between the machine and the environment, and (b) it views proofs as a winning strategy of the machine. From this game-based viewpoint, a poweful heuristic can be extracted and a fair degree of determinism in proof search can be obtained. This article proposes a new deductive system LKg with respect to first-order logic and proves its soundness and completeness.
key words: *proof procedures, heuristics, game semantics, classical logic*

1. Introduction

The Gentzen sequent system LK plays a key role in modern theorem proving. Unfortunately, the LK system and its variants based on focused proof [1] (as well as resolution and tableux (see [7] for discussions)) are typically based on blind search and, therefore, does not provide the best strategy if we want a short proof.

In this paper, inspired by the seminal work of [3], we present a variant of LK, called LKg (g for game), which yields a proof in normal form with the following features:

- All the quantifier inferences are processed first. This is achieved via deep inference.
- If there are several quantifiers to resolve in the sequent, we apply to sequents a technique called *stability analysis*, a powerful heuristic technique which greatly cuts down the search space for finding a proof.
- A quantifier is processed only when it needs to be processed. In other words, our terminating condition (our axiom) is much better than the traditional proof procedures including [1].

In essence, LKg is a *game-viewed* proof which captures *game-playing* nature in proof search. It views

1. sequents as games between the machine and the environment,
2. proofs as a winning strategy of the machine, and
3. $\forall xF$ as the env's move and $\exists xF$ as the machine's move.

At each stage, we construct a proof by the following rules:

1. If the sequent is stable, then it means that the machine is the current winner. In this case, it requests the user to make a move.
2. If the sequent is instable, then it means that the environment is the current winner. In this case, the machine makes a move.

In this way, a fair (probably maximum) degree of determinism can be obtained from the LKg proof system.

In this paper we present the proof procedure for first-order classical logic. The remainder of this paper is structured as follows. We describe LKg in the next section. In Sect. 3, we present some examples of derivations. In Sect. 4, we prove the soundness and completeness of LKg. Section 5 concludes the paper.

2. The Logic LKg

The formulas are the standard first-order classical formulas, with the features that (a) $\top$, $\bot$ are added, and (b) $\neg$ is only allowed to be applied to atomic formulas. Thus we assume that formulas are in negation normal form.

The deductive system LKg below axiomatizes the set of valid formulas. LKg is a one-sided sequent calculus system, where a sequent is a multiset of formulas. Our presentation closely follows the one in [3].

First, we need to define some terminology.

1. A **surface occurrence** of a subformula is an occurrence that is not in the scope of any quantifiers ($\forall$ and/or $\exists$).
2. A sequent is **propositional** iff all of its formulas are so.
3. The **propositionalization** $\|F\|$ of a formula F is the result of replacing in F all $\exists$-subformulas by $\bot$, and all $\forall$-subformulas by $\top$. The **propositionalization** $\|F_1, \ldots, F_n\|$ of a sequent $F_1, \ldots, F_n$ is the propositional formula $\|F_1\| \vee \ldots \vee \|F_n\|$.
4. A sequent is said to be **stable** iff its propositionalization is classically valid; otherwise it is **unstable**.
5. The notation $F[E]$ repesents a formula F together with some surface occurrence of a subformula E.

The rules of LKg

LKg has the five rules listed below, with the following additional conditions:

1. X:stable means that X must meet the condition that it is stable. Similarly for X:unstable.

Manuscript received April 26, 2019.
Manuscript revised August 12, 2019.
Manuscript publicized November 21, 2019.
[†]The author is with Department of Computer Eng., DongA University, Korea.
a) E-mail: khkwon@dau.ac.kr
DOI: 10.1587/transinf.2019FCL0003

2. Γ is a multiset of formulas and F is a formula.
3. In $\exists$-Choose, t is a closed term, and $H(t)$ is the result of replacing by t all free occurrences of x in $H(x)$.
4. The 'Succ' rule reads as follows: A stable sequent X containing no surface occurrence of $\forall xH(x)$ is derivable.
5. The 'Fail' rule reads: An unstable sequent X containing no surface occurrence of $\exists xH(x)$ is not derivable.

Fail

$$\frac{\bot}{X\text{:unstable}}\ (\text{no surf } \exists xH(x) \text{ in } X)$$

$\exists$-Choose

$$\frac{\Gamma, F[H(t)]}{\Gamma, F[\exists xH(x)]\text{:unstable}}$$

Replicate

$$\frac{\Gamma, F[\exists xH(x)], F[\exists xH(x)]}{\Gamma, F[\exists xH(x)]\text{:unstable}}$$

Succ

$$\frac{\top}{X\text{:stable}}\ (\text{no surface } \forall xH(x) \text{ in } X)$$

$\forall$-Choose

$$\frac{\Gamma, F[H(\alpha)]}{\Gamma, F[\forall xH(x)]\text{: stable}}(\alpha \text{ is a new constant})$$

In the above, the "Replicate" rule is an optimized version of what is known as Contraction, where contraction occurs only when there is a surface occurrence of $\exists xH(x)$.

A **LKg-proof** of a sequent X is a sequence $X_1, \ldots, X_n$ of sequents, with $X_n = X$, $X_1 = \top$ such that, each X_i follows by one of the rules of LKg from X_{i-1}.

3. Examples

Below we describe some examples.

Example 3.1: *The formula $\forall x \exists y\big(p(x) \to p(y)\big)$ is provable in LKg as follows:*

1. $p(\alpha) \to p(\alpha)$ *Succ*
2. $\exists y\big(p(\alpha) \to p(y)\big)$ *$\exists$-Choose*
3. $\forall x \exists y\big(p(x) \to p(y)\big)$ *$\forall$-Choose*

Example 3.2: *The formula $\exists y \forall x\big(p(x) \to p(y)\big)$ is provable in LKg as follows:*

1. $\big(p(\alpha_1) \to p(a)\big), \big(p(\alpha_2) \to p(\alpha_1)\big)$ *Succ*
2. $\big(p(\alpha_1) \to p(a)\big), \forall x\big(p(x) \to p(\alpha_1)\big)$ *$\forall$-Choose*
3. $\big(p(\alpha_1) \to p(a)\big), \exists y \forall x\big(p(x) \to p(y)\big)$ *$\exists$-Choose*
4. $\forall x\big(p(x) \to p(a)\big), \exists y \forall x\big(p(x) \to p(y)\big)$ *$\forall$-Choose*
5. $\exists y \forall x\big(p(x) \to p(y)\big), \exists y \forall x\big(p(x) \to p(y)\big)$ *$\exists$-Choose*
6. $\exists y \forall x\big(p(x) \to p(y)\big)$ *Replicate*

On the other hand, the formula $\big(\exists xp(x) \to \forall yp(y)\big)$ which is invalid can be seen to be unprovable. This can be derived only by two $\forall$-Choose rules and then the premise should be of the form $\neg p(\alpha_1), p(\alpha_2)$ for some new constants α_1, α_2. The latter is not classically valid.

We conclude this section by discussing performance aspects of LK and LKg. LK (and its variants) processes $\exists$,$\forall$-quantifiers in an eagerly fashion, even when it is unnecessary. On the other hand, LKg processes these quantifiers in a lazy fashion, only when it is absolutely necessary. It can be expected then that LKg typically has shorter proofs for validity (and invalidity if any) than LK.

As an example, consider the following:
$p(a) \land \forall x_1, \ldots, \forall x_n q(x_1, \ldots, x_n)$.
LKg immediately declares that this formula is invalid, as it is unstable and there is no surface occurrence of $\exists xH(x)$. In contrast, LK typically processes $\forall x_1, \ldots, \forall x_n$, which is redundant.

As a second example, consider the following:
$((p \to q) \land (q \to r)) \to (p \to (r \lor \exists x_1, \ldots, \exists x_n s(x_1)))$.
Again, LKg immediately declares that this formula is valid, as it is stable and there is no surface occurrence of $\forall xH(x)$. In contrast, LK is likely to process $\exists x_1, \ldots, \exists x_n$, which is redundant.

4. The Soundness and Completeness of LKg

We now present the soundness and completeness of LKg.

Theorem 4.1: 1. If LKg terminates with success for X, then X is valid.
2. If LKg terminates with failure for X, then X is invalid.
3. If LKg does not terminate for X, then X is invalid.

Proof. Consider an arbitrary sequent X.

Soundness: Induction on the length of derivatons.

Case 1: X is derived from Y by $\exists$-Choose. By the induction hypothesis, Y is valid, which implies that X is valid.

Case 2: X is derived from Y by Replicate. By the induction hypothesis, Y is valid. Then, it is easy to see that X is valid.

Case 3: X is derived from Y by Succ.

In this case, we know that there is no surface occurrences of $\forall xH(x)$ in X and $||X||$ is classically valid. It is then easy to see that, reversing the propositionalization of $||X||$ (replacing $\bot$ by any formula of the form $F[\exists xH(x)]$) preserves validity. For example, if X is $p(a) \to p(a), \exists xq(x)$, then $\|X\|$ is valid and X is valid as well.

Case 4: X is derived from Y by $\forall$-Choose.

Thus, there is an occurrence of $\forall x H(x)$ in X. The machine makes a move by picking up some fresh constant c not occurring in X. Then, by the induction hypothesis, the premise is valid. Now consider any interpretation I that makes the premise true. Then it is easy to see that the conclusion is true in I. It is commonly known as "generalization on constants".

Completeness: Assume LKg terminates with failure.

We proceed by induction on the length of derivations.

If X is stable, then there should be a LKg-unprovable sequent Y with the following condition.

Case 1: $\forall$-Choose: X has the form $\Gamma, F[\forall x G(x)]$, and Y is $\Gamma, F[G(\alpha)]$, where α is a new constant not occurring in X. In this case, Y is a LKg-unprovable sequent, for otherwise X is LKg-provable. By the induction hypothesis, Y is not true in some interpretation I. Then it is easy to see that X is not true in I. Therefore X is not valid.

Next, we consider the cases when X is not stable. Then there are three cases to consider.

Case 2.1: Fail: In this case, there is no surface occurrence of $\exists x G(x)$ and the algorithm terminates with failure. As X is not stable, $\|X\|$ is not classically valid. If we reverse the propositionalization of $\|X\|$ by replacing $\top$ by any formula with some surface occurrence of $\forall G(x)$, we observe that invalidity is preserved. Therefore, X is not valid.

Case 2.2: $\exists$-Choose: In this case, X has the form $\Gamma, F[\exists x G(x)]$, and $Y(t)$ is $\Gamma, F[G(t)]$, where t is a closed term. In this case, $Y(t)$ is a LKg-unprovable sequent for any t, for otherwise X is LKg-provable. By the induction hypothesis, none of $Y(t)$ is valid and thus none of $Y(t)$ is not true in some interpretation I. Then it is easy to see that X is not true in I. Therefore X is not valid.

Case 2.3: Replicate: In this case, X has the form $\Gamma, F[\exists x G(x)]$, and Y is $\Gamma, F[\exists x G(x)], F[\exists x G(x)]$. In this case, Y is a LKg-unprovable sequent, for otherwise X is LKg-provable. By the induction hypothesis, Y is not valid and is not true in some interpretation I. Then it is easy to see that X is not true in I. Therefore X is not valid.

Now assume LKg is not terminating, because Replicate occurs infinitely many times. We prove this by contradition.

Assume that X is valid but unprovable. Let Z be an infinite multiset of propositional formulas obtained by applying infinite numbers of Replicate, together with $\exists$-Choose and $\forall$-Choose rules. Then it is easy to see that Z remains still valid but unprovable. By the compactness theorem on propositional logic, there is a finite subset Z' of Z, which is a valid sequent. Then there must be a step t in the procedure such that, after t, Z' is derived. Then it is easy to see that Z' remains LKg-unprovable. However, as Z' is valid, it must be provable by Succ. This is a contradiction and, therefore, X is not valid. ■

5. Some Optimizations

Although LKg performs well for valid sequents, it performs poorly for invalid sequents. For example, it does not even terminate for the invalid sequent $p(a), p(b) \wedge \exists x q(x)$.

For this reason, what we need is a good heuristic for determining, in a simple yet effective way, whether a sequent is invalid. In this section, we employ a simple heuristic called *maximum propositionalization* which replaces in a sequent X all $\exists$-subformulas by $\top$. If X' is obtained from X by maximum propositionalization, then it is easy to observe that if X' is invalid, then X is invalid as well.

The deductive system LKg$'$ uses this heuristic. First, we need to define some terminology.

1. The **max-propositionalization** $\|F\|_{max}$ of a formula F is the result of replacing in F all $\exists$-subformulas by $\top$, and all $\forall$-subformulas by $\top$. This process naturally extends to sequents.
2. The **min-propositionalization** $\|F\|_{min}$ of a formula F is the result of replacing in F all $\exists$-subformulas by $\perp$, and all $\forall$-subformulas by $\perp$. This process naturally extends to sequents.
3. A sequent is said to be **max-p-invalid** iff its max-propositionalization is classically invalid. A sequent is said to be **min-p-valid** iff its min-propositionalization is classically valid.

The rules of LKg$'$

Below, X:stable means that X is stable but not min-p-valid. stable. Similarly X:unstable means that X is unstable but not min-p-valid.

Fail

$$\frac{\perp}{X\text{: max-p-invalid}}$$

$\exists$-Choose

$$\frac{\Gamma, F[H(t)]}{\Gamma, F[\exists x H(x)]\text{:unstable}}$$

Replicate

$$\frac{\Gamma, F[\exists x H(x)], F[\exists x H(x)]}{\Gamma, F[\exists x H(x)]\text{:unstable}}$$

Succ

$$\frac{\top}{X\text{: min-p-valid}}$$

∀-Choose

$$\frac{\Gamma, F[H(\alpha)]}{\Gamma, F[\forall x H(x)]\text{: stable}} \ (\alpha \text{ is a new constant})$$

The heuristic employed in LKg′ is quite simple and needs to be improved. For example, it does not apply well to the invalid sequent $p(a), \exists x p(x)$. It would be nice to improve our heuristic so that it can apply to a wider class of invalid sequents.

Acknowledgments

This work was supported by Dong-A University Research Fund.

References

[1] J.-M. Andreoli, "Logic programming with focusing proofs in linear logic," Journal of Logic and Computation, vol.2, no.3, pp.297–347, 1992.

[2] G. Gentzen, "Investigations into Logical Deduction," The Collected Papers of Gerhard Gentzen, pp.68–131, 1969.

[3] G. Japaridze, "Introduction to computability logic," Annals of Pure and Applied Logic, vol.123, no.1-3, pp.1–99, 2003.

[4] G. Japaridze, "From truth to computability I," Theoretical Computer Science, vol.357, no.1-3, pp.100–135, 2006.

[5] G. Japaridze, "Computability logic: A formal theory of interaction," ed. D. Goldin, S. Smolka, and P. Wegner, Interactive Computation: The New Paradigm, pp.183–223, Springer, 2006.

[6] G. Japaridze, "In the beginning was game semantics," ed. O. Majer, A.-V. Pietarinen, and T. Tulenheimo, Games: Unifying Logic, Language, and Philosophy, pp.249–350, Springer, 2009.

[7] S. Reeves and M. Clarke, Logic for Computer Science, Addison Wesley, 1990.