Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 212 (2008) 87–102
www.elsevier.com/locate/entcs

Proof Search and Counter Model of Positive Minimal Predicate Logic

Tao Xue[1,2] Qichao Xuan[3]

State Key Laboratory of Computer Science
Institute of Software Chinese Academy of Science
Beijing, China

Abstract

This paper presents an algorithm of proof search for positive formulas in minimal predicate logic. It is based on the LJB deduction system introduced in[2]. The algorithm returns a deduction tree, and hence a proof, when the formula is provable, and a counter-model will be constructed when the formula is unprovable. The soundness and the completeness are proved.

Keywords: Kripke model, LJB, counter model, proof search

1 Introduction

To find a proof for a logical formula is a classical subject in mathematic logic, and proof search is a method with a long history to deal with the problem. It is used in a variety of system of logic, including classical, intuitionistic, linear and modal systems, at the propositional, first- and higher-order levels. There are many researches focusing on constructive logics like intuitionisitc logic, mainly because of the researches in type theory and the relation between intuitionistic logic and type theory through Curry-Howard's isomorphism.

Although proof-search is a good method to find out the proof, when a failure comes, we lack an intuition view for the unprovable. However, Kripke model can be a good counter model to explain the unprovable more intuitively, and usually, the trace of the failure contains enough information for the counter model construction. Sequent calculus with multiple conclusions is a standard method to construct

[1] This work is partially supported by NSFC 60673045, NSFC major research program 60496321 and NSFC 60721061.

[2] Email: xuet@ios.ac.cn

[3] Email: xuanqichao-1982@163.com

doi:10.1016/j.entcs.2008.04.055

counter-model, and there has been many works on it. In [3] Roy Dyckhoff gives a contraction-free sequent calculus LJT in which proof search does not loop. Then Roy Dyckhoff and Pinto used a variant of LJT and defined a new calculus CRIP to capture unprovablity in [6]. They used LJT to generate proof and CRIP to construct counter-model. In [8], the system Porgi can return either a proof deduction or a Kripke counter model, but it also used LJT to get the proof and CRIP to construct counter model, so it still needed to deal the provable and unprovable in separate algorithm. Hirokawa and Nagano[5] showed a different method to get proof or counter-model in long normal form proof, they used only one algorithm for both proof and counter-model generating. Moreover, this algorithm showed an that the structure of counter-model is in accordance with the structure of proof deduction.

All the works up are in propositional logic, and we find seldom on predicate logic. Mainly because the provability of the logic system been considered is required, but provability in minimal predicate logic is not decidable. However in [2], Dowek and Jiang proved the decidability of positive fragment of minimal predicate logic and proposed a proof search algorithm base on special sequent calculus called LJB.

In this paper, we use the LJB deduction system, and extent the algorithm of constructing counter model in [5]. Given a closed positive formula in minimal predicate logic, we introduce an algorithm to get proofs to the formulas which are provable and get counter-models with Kripke structure to the formulas which are not provable. Like in [5], our algorithm shows a relation between the structure of proof and structure of counter model. We don't consider the efficiency and a loop-checking is needed.

Our algorithm firstly search the proof backward using the rules in the LJB system. If the formula is provable, the search will finally get a proof. If it is unprovable, the search will stop in a stage where we can not use any rule to search upwards, or we find a loop. In this case, we start to construct the counter-model from the leaves to the root. We assign every variable a different constant, give the atomic formulas of every leaf an original assignment, and construct model with the subnodes for every internal node.

Section 2 will give some basic knowledge for the whole paper. In section 3, the algorithm of proof search and counter-model construction is introduced, and two small examples are given. In section 4, we prove the correctness of the counter model construction. Finally we give a proof to the soundness and completeness in the last section.

2 Preliminaries

In this section, we call the definitions and some properties of positive formula, LJB deduction system in [2], and the Kripke model in [7]. There are some differences in notation and presentation. Further details can be got in the references.

2.1 Positive formula

In minimal predicate logic, the syntax of terms and formulas are given by

$$t = x \mid f(t, \ldots, t)$$

$$A = P(t, \ldots, t) \mid (A \to A) \mid \forall x A$$

Superfluous parentheses are omitted as usual. Free and bound occurrences of variables in a formula are defined as usual.

A context is a finite multiset of formula. A sequent $\Gamma \vdash A$ is a pair formed by a context Γ and a formula A.

Definition 2.1 (Free and bound variables of a context) Free and bound variables of a context are defined by

- $FV(\{A_1, \ldots, A_n\}) = FV(A_1) \cup \ldots \cup FV(A_n)$.
- $BV(\{A_1, \ldots, A_n\}) = BV(A_1) \cup \ldots \cup BV(A_n)$.

A formula in minimal predicate logic is positive if all its universal quantifier occurrences are positive. More precisely, the set of positive and negative formula are define by induction as follows.

Definition 2.2 (Positive and negative formulas and sequents)

- An atomic formula is positive and negative,
- a formula of the form $A \to B$ is positive(resp. negative) if A is negative(resp. positive) and B is positive(resp. negative),
- a formula of the form $\forall x A$ is positive if A is positive,
- a sequent $A_1, \ldots, A_n \vdash B$ is positive if $A_1, \ldots, A_n$ are negative and B is positive.

Note 1 *A negative formula has the form $A_1 \to \ldots \to A_n \to P$ where P is an atomic formula, $A_1, \ldots, A_n$ are positive formulas, and we note $T(A) = P$*

2.2 LJB: a sequent calculus with brackets

When we have a sequent of the form $\Gamma \vdash \forall x A$, we usually need to rename the variable x with a variable x' that is free neither in Γ nor in A in order to apply the $R\forall$ rule. But here we take another sequent calculus introduced in [2], where, instead of renaming the variable x ,we bind x and all the other variables bound in A, in the context Γ with brackets. And obtain the sequent $[\Gamma]_V \vdash A$, V is the set of all bound variables in $\forall x A$.

Definition 2.3 (LJB-contexts and items)LJB-contexts and items are mutually inductively defined as follows.

- A LJB-context Γ is a finite multiset of items $I_1, \ldots, I_n$,
- an item I is either a formula or an expression of the form $[\Gamma]_V$ where V is a set of vriables and Γ is a context.

In the item $[\Gamma]_V$ the variables of V are bound by the symbol [ ].

Definition 2.4 (Free and bound variables of a LJB-context and of an item) The set of free variables of a LJB-context is defined by

- $FV(\{I_1, \ldots, I_n\}) = FV(I_1) \cup \ldots \cup FV(I_n)$,
  and the set of free variables of an item by
- $FV(A) = FV(A)$,
- $FV([\Gamma]_V) = FV(\Gamma) \backslash V$.

the set of bound variables of a LJB-context is defined by

- $BV(\{I_1, \ldots, I_n\}) = BV(I_1) \cup \ldots \cup BV(I_n)$,
  and the set of free variables of an item by
- $BV(A) = BV(A)$,
- $BV([\Gamma]_V) = BV(\Gamma) \cup V$.

A LJB-sequent $\Gamma \vdash A$ is a pair formed by a LJB-context Γ and a formula A.

The system LJB is formed by two sets of rules: the usual deduction rules and additional transformation rules. The transformation rules deal with bracket manipulation. They form a terminating rewrite system. The first transformation rule allows to replace an item of the form $[I, \Gamma]_V$ by two items I and $[\Gamma]_V$ provided no free variable of I is in V. The second rule allows to remove trivial items. The third rule to replace two identical item by one.

Definition 2.5 (Cleaning LJB-contexts) We consider the following rules simplifying LJB-contexts, where I is a item and Γ is a LJB-context.

- $[I, \Gamma]_V \longrightarrow I, [\Gamma]_V, \quad if\ FV(I) \cap V = \emptyset$
- $[\ ]_V \longrightarrow \emptyset$
- $II \longrightarrow I$

As usual the rules may be applied anywhere in a LJB-context.

Proposition 2.6 (Termination) *The rewrite system of Definition 2.5 terminates.*

Note 2 *: We take $\Gamma \downarrow$ as the normal form of a context Γ with the rules.*

Definition 2.7 (LJB , A sequent calculus with brackets)

$$\frac{\Gamma' \vdash A_1 \quad \ldots \quad \Gamma' \vdash A_n}{\Gamma \vdash P} L \rightarrow$$

where

$$\Gamma = \Gamma_1, [\Gamma_2, [\ldots \Gamma_{i-1}, [\Gamma_i, A_1 \rightarrow \ldots \rightarrow A_n \rightarrow P]_{V_{i-1}} \ldots]_{V_2}]_{V_1}$$
$$\Gamma' = ([\ldots [[\Gamma_1]_{V_1}, \Gamma_2]_{V_2}, \ldots, \Gamma_{i-1}]_{V_{i-1}}, \Gamma_i, A_1 \rightarrow \ldots \rightarrow A_n \rightarrow P) \downarrow$$

P is atomic and has no free variable in $V_1 \cup V_2 \cup \ldots \cup V_{i-1}$

$$\frac{[\Gamma]_V \downarrow \vdash A}{\Gamma \vdash \forall x A} R\forall$$

where V is the set of all variables bound in $\forall xA$

$$\frac{(\Gamma, A)\downarrow \vdash B}{\Gamma \vdash A \to B} R \to$$

2.3 Kripke model

Definition 2.8 (Kripke frame) A Kripke frame for intuitionistic predicate logic is a triple $< W, R, D >$, where W is a non-empty set, R is a binary reflexive and transition relation on W, D is a domain function assigning to every $w \in W$ a non-empty set $D(w)$ expanding with respect to R: wRw' implies $D(w) \subseteq D(w')$.

Every world w of a Kripke frame $< R, W, D >$ corresponds an extension L_w of the language of predicate logic obtained by adding constants for all elements of the domain $D(w)$. We identify such a constant with the corresponding element of $D(w)$. Hence sentences of L_w are formulas containing no free occurrences of variables but possibly containing objects of $D(w)$.

Definition 2.9 (Kripke model) A Kripke model for intuitionistic predicate logic is a 4-tuple $< W, R, D, V >$, where $< W, R, D >$ is a Kripke frame and V assigns a function $V(f)$ to every function symbol f and a predicate symbol $V(P)$ to every predicate P, and the following condition are satisfied.
For all $w \in W$, $d_1, \ldots, d_n \in D(w)$ (and n-ary f,P):
(1) $V(f)(d_1, \ldots, d_n, w) \in D(w)$ and $V(P)(d_1, \ldots, d_n, w) \in \{0, 1\}$
(2) wRw' implies $V(f)(d_1, \ldots, d_n, w') = V(f)(d_1, \ldots, d_n, w)$
(3) wRw' and $V(P)(d_1, \ldots, d_n, w) = 1$ implies $V(P)(d_1, \ldots, d_n, w) = 1$

Definition 2.10 (values in Kripke model)
(1) The value $V(t, w) \in D(w)$ of a constant term $t \in D(w)$:
Constants: $V(d, w) = d$ if $d \in D(w)$
Composite terms: $V(f(t_1, \ldots, t_p), w) = V(f)(V(t_1, w), \ldots, V(t_p, w), w)$
(2) The value $V(A, w) \in \{0, 1\}$ for a formula A in w.
$V(P(t_1, \ldots, t_n), w) = V(P)(V(t_1, w), \ldots, V(t_n, w), w)$
$B \to C : V(B \to C, w) = 1$ iff for all w' with wRw' we have $V(B, w') = 1$ implies $V(C, w') = 1$.
$\forall xB : V(\forall xB) = 1$ iff for all w' with wRw' we have $V(B[x/d], w') = 1$ for all $d \in D(w')$.

We say a formula A is valid or $\models A$, if for every model $< W, R, D, V >$, every $w \in W$, and every substitution $\sigma = [x_1/d_1, \ldots, x_n/d_n]$ of objects in $D(w)$ for free variables of A, $V(A\sigma, w) = 1$.

Proposition 2.11 (Monotonicity) *Assume wRw', then $V(A, w) = 1$ implies $V(A, w') = 1$.*

Definition 2.12 (Pointed frame and pointed model) A pointed frame is an ordered 4-tuple $< G, W, R, D >$, where $< W, R, D >$ is a Kripke frame, $G \in W$ and GRw for all $w \in W$. A pointed model is a tuple M=$< G, W, R, D, V >$ where

$< G, R, D, V >$ is a pointed frame and V is a valuation on $< W, R, D >$. Truth in M is truth in the world G: $M \models A$ iff $V(G, A) = 1$

Definition 2.13 (Partial order) A binary relation R on a set W is a partial order iff R is reflexive, transitive and antisymmetric.

Proposition 2.14 *A formula is valid iff it is true in all pointed models, iff it is true in all pointed models where accessibility relation is a partial order.*

Note 3 *For a easier writing, in the remain of the article we will use* $\leq$ *instead of* R*, if* wRw' *then we write* $w \leq w'$ *or* $w' \geq w$*. We also write* $w \models A$ *instead of* $V(A, w) = 1$*, and* $w \not\models A$ *instead of* $V(A, w) = 0$*.*

Here we have some other definitions for our model construction

Definition 2.15 (Support sets) Given a Kripke model $< W, R, D, V >$ and a node $w \in W$, then *Support* assigns a set of atomic formulas to every node w as following $Support(w) = \{A \mid w \models A,\ A \text{ is atomic}\}$. And $Support(W) = \{A \mid w \models A,\ A \text{ is atomic}\ ,\ \forall w \in W\}$.

Definition 2.16 (Singleton model) A singleton model M with a set of atomic formulas $S = \{A_1, \ldots, A_n\}$ and domain D_0 is a Kripke model $< W, R, D, V >$ such that $W = \{w\}$ is a singleton set, R is the trivial order wRw, $D(w) = D_0$, and Support(w)=S.

Definition 2.17 (Joint model) Given Kripke models $M_i =< W_i, R_i, D_i, V_i >, i \in \{1, \ldots, n\}$, the joint model of M_i with respect to Σ is a model $M =< W, R, D, V >$ such that
(1) $W = \{w_0\} \cup W_1 \cup \ldots \cup W_n$,
(2) $w_0 R u, \forall u \in W$,
(3) $D = D_1 \cap \ldots \cap D_n$,
(4) $Support(w_0) = Support(W_1) \cap \ldots \cap Support(W_n) - \Sigma$.
Σ is a set of atomic or an $\emptyset$.

3 Proof search algorithm and counter model construction

In this section we show a proof search algorithm which gives a proof of a given closed positive formula E in LJB when it is provable, or construct an counter model when E is not provable.

First, to the given closed positive formula E, our aim is to search a proof for $\vdash E$. More generally, we consider the search for a LJB sequent $\Gamma \vdash P$, and we use a list of sequent ξ to store the search path for loop checking. The algorithm *Search* takes a LJB sequent $\Gamma \vdash P$ and ξ as parameters, and gives a trees of LJB sequents as a result. The tree is called a *deduction tree* with leaves labeled in three kinds, "axiom", "stop" and "loop". If one leaf is label by "axiom", it implies all the leaves are labeled by "axiom", then the tree is a proof of the sequent and called *proof tree*.

Else we will get a deduction tree with leaves labeled "stop" or "loop", and we call it a *fail tree*.

More precisely, we explain how the *deduction tree* is constructed step by step. A *deduction tree* is composed inductively by the rules of LJB system, we just make a explanation of rule $L \rightarrow$ since the cases of other two rules are trivial. Given a sequent $\Gamma \vdash P$, P is atomic, there are several choices to use the rule $L \rightarrow$(fig.1.a). If there is at least one provable case, suppose case i, then we take all the nodes in this case as the subnodes of $\Gamma \vdash P$(fig.1.b). If all the cases are unprovable, that means there is at least one node unprovable in every case. Then from each case we take an unprovable node, and make all these nodes to be subnodes of $\Gamma \vdash P$(fig.1.c). Finally, when we come to a node which we can not use any rule on the sequent, we label it "stop". Or, when we look up ξ and find the sequent been dealt before, we label it "loop". If the sequent can be proved trivially by $L \rightarrow$, it will be labeled "axiom".

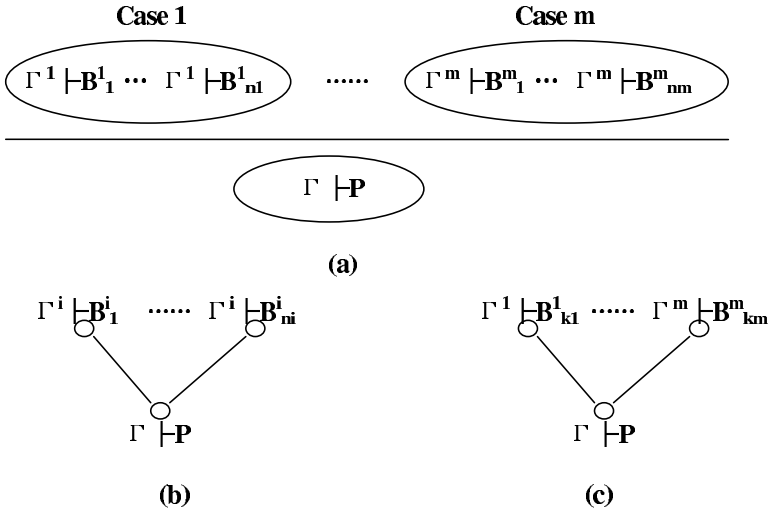

Fig. 1.

Definition 3.1 (Search algorithm) $Search(\Gamma \vdash P; \xi)$ is defined inductively as follows:

1. If P is atomic.

1.1. If $P \in \Gamma$. Then it's a proof and return a node $P \in \Gamma$ labeled "axiom".

1.2. If $P \notin \Gamma$.

1.2.1. If $\Gamma \vdash P \in \xi$. Then it's not a proof and return a node $\Gamma \vdash P$ labeled "loop".

1.2.2. If $\Gamma \vdash P \notin \xi$.

1.2.2.1. If Γ does not contain any such formula A, which is bracketed by V and $T(A) = P$, P has no free variables in V. Then it's not a proof and return $\Gamma \vdash P$ labeled "stop".

1.2.2.2. If Γ contains some A_i bracketed by V_i st. $T(A_i) = P$, P has no free variables in V_i. Let $A_1, \ldots, A_m$ be all of such formulas and $A_i = B_1^i \rightarrow \ldots \rightarrow B_{n_i}^i \rightarrow P (1 \leq i \leq m)$. Then for every A_i, Γ can be written as

$\Gamma = \Gamma_1^i, [\Gamma_2^i, [\ldots \Gamma_{n_i-1}^i, [\Gamma_{n_i}^i, B_1^i \to \ldots \to B_{n_i}^i \to P]_{V_{n_i-1}^i} \ldots]_{V_2^i}]_{V_1^i}$
and we have
$\Gamma^i = ([\ldots [[\Gamma_1^i]_{V_1^i}, \Gamma_2^i]_{V_2^i}, \ldots \Gamma_{n_i-1}^i]_{V_{n_i-1}^i}, \Gamma_{n_i}^i, B_1^i \to \ldots \to B_{n_i}^i \to P) \downarrow$
Then by using the algorithm recursively, we have
$u_k^i = Search(\Gamma^i \vdash B_k^i \,; \Gamma \vdash P + \xi)$ for $1 \le i \le m$, $1 \le k \le n_i$.
1.2.2.2.1. If for some i $(1 \le i \le m)$, all $u_1^i, \ldots, u_{n_i}^i$ are proofs. Return a proof

$$\frac{u_1^i \quad \ldots \quad u_{n_i}^i}{\Gamma \vdash P}$$

1.2.2.2.2. If for each i $(1 \le i \le m)$, we have some $u_{k_i}^i (1 \le k_i \le n_i)$ is not a proof. Return a tree whose root is $\Gamma \vdash P$ and has all the $u_{k_i}^i$ as subtrees above the root.
2. If $P = \forall x A$. We have $u = Search([\Gamma]_V \downarrow \vdash A, \Gamma \vdash P)$.
2.1. If u is a proof, return a proof

$$\frac{u}{\Gamma \vdash P}$$

2.2. If u is not a proof, return a tree whose root is $\Gamma \vdash P$ and has u as subtree above the root.
3. If $P = A \to B$. We have $u = Search(([\Gamma_]V, A) \downarrow \vdash B, \Gamma \vdash P)$.
3.1. If u is a proof, return a proof

$$\frac{u}{\Gamma \vdash P}$$

3.2. If u is not a proof, return a tree whose root is $\Gamma \vdash P$ and has u as subtree above the root.

Example 3.2 We try to prove the formula

$$\forall x((P(x) \to Q(x)) \to \forall y(Q(y) \to P(x) \to Q(x)))$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(x) \to Q(x), Q(y), P(x) \vdash P(x)}\, L\to}{P(x) \to Q(x), Q(y), P(x) \vdash Q(x)}\, L\to}{P(x) \to Q(x) \vdash Q(y) \to P(x) \to Q(x)}\, R\to (2)}{P(x) \to Q(x) \vdash \forall y(Q(y) \to P(x) \to Q(x))}\, R\forall}{\vdash (P(x) \to Q(x)) \to \forall y(Q(y) \to P(x) \to Q(x))}\, R\to}{\vdash \forall x((P(x) \to Q(x)) \to \forall y(Q(y) \to P(x) \to Q(x)))}\, R\forall$$

Thus, the formula is provable.

If $Search(\vdash E, nil)$ returns not a proof but a *fail tree* t for a given closed positive formula E, we need to construct a counter model. For a counter model, we need only

to get one evidence for uprovable, so we give an assignment to every free variable. For the sake of our assignment, we define a *c-substitute* to substitute all the free variables in the nodes of the *fail tree* t with constants from root to leaves, and we assign the variables with fresh constants instead of using brackets so that we can move all the brackets away. The detail is followed.

Definition 3.3 (c-substitute) For the fail tree t, $\Gamma \vdash P$ is a node of the tree we define the *c-substitute* of the node by induction on structure.
1. If $\Gamma \vdash P$ is the root of t.
The *c-substitute* of this node is $\Gamma \vdash P$.
2. If $\Gamma \vdash P$ is a subnode of node $\Gamma_0 \vdash A$ which has *c-substitute* $\Sigma \vdash A'$
2.1. If A is atomic.
Then Γ and Γ_0 can be written as:
$\Gamma = ([\ldots[[\Gamma_1]_{V_1}, \Gamma_2]_{V_2}, \ldots, \Gamma_{i-1}]_{V_{i-1}}, \Gamma_i, B_1 \to \ldots \to B_i \to A) \downarrow$
$\Gamma_0 = \Gamma_1, [\Gamma_2, [\ldots \Gamma_i, [\Gamma_i, B_1 \to \ldots \to B_n \to A]_{V_{i-1}} \ldots]_{V_2}]_{V_1}$
and $P = B_j$, $j \in \{1, \ldots, n\}$.
In $\Sigma \vdash A'$, B_j has been substituted by P', then the *c-substitute* of $\Gamma \vdash P$ is $\Sigma \vdash P'$
2.2. If $A = \forall x P$.
A' can be expressed as $\forall x' P'$. Let a be a fresh constant which is not in $\Sigma \vdash A'$, the *c-substitute* of $\Gamma \vdash P$ is $\Sigma \vdash P'(x/a)$.
2.3. If $A = Q \to P$
A' can be expressed as $Q' \to P'$, the *c-substitute* of $\Gamma \vdash P$ is $\Sigma, Q' \vdash P'$.

Next, we come to construct a Kripke model. We start from leaves, and transform them into singleton models. At each inner node, we construct a joint model with the submodels obtained from the subtrees of the node. When there is a leaf of sequent $\Gamma \vdash P$ labeled with "loop", the tree contains another occurrence of $\Gamma \vdash P$ below the leaf, then we identify all the nodes between the two. Given such a deduction tree t, the following algorithm shows how to generate a model t^m as well as the set t^l of sequences to tree repetition.

Definition 3.4 (construction of Kripke model, t^m and t^l)
Given a closed positive formula E, $t = Search(\vdash E, nil)$ is a fail tree, we do *c-substitute* to all nodes in t, and suppose Con(E) is a set of all constants we have used in the *c-substitute*.
Suppose $\Gamma \vdash P$ is a node in t, and the *c-substitute* of $\Gamma \vdash P$ is $\Sigma \vdash P'$.
1. If P is atomic.
1.1. If t is a leaf labeled "loop" with $\Gamma \vdash P$. t^m is singleton model with $T(\Sigma) \backslash \{P'\}$, and has Con(E) as its domain, $T(\Sigma) = \{T(A) | A \in \Sigma\}$, $t^l = \Gamma \vdash P$.
1.2. If t is a leaf labeled "stop" with $\Gamma \vdash P$. t^m is singleton model with $T(\Sigma)$, and has Con(E) as its domain, $t^l = \emptyset$.
1.3. If t is the root with subtrees $t_1, \ldots, t_n$.
1.3.1. If $\Gamma \vdash P$ does not occur in any t_i^l, then t^m is the joint model of $t_1^m, \ldots t_n^m$ with respect to $\{P'\}$, $t^l = t_1^l \cup \ldots \cup t_n^l$.
1.3.2. If $\Gamma \vdash P$ occurs in some t_i^l, then construct a model M which is the joint model of $t_1^m, \ldots, t_n^m$ with respect to $\{P'\}$, t^m is obtained from M by identifying all

the nodes r which satisfy $\Gamma \vdash P \in r^l$ in t_i^m. $t^l = t_1^l \cup \ldots \cup t_n^l \backslash \{\Gamma \vdash P\}$.
2. If $P = \forall x A$, then t has a subtree t_1. t^m is a joint model of t_1^m with respect to $\emptyset$, and $t^l = t_1^l$.
3. If $P = A \to B$, then t has a subtree t_1. t^m is a joint model of t_1^m with respect to $\emptyset$, and $t^l = t_1^l$.

For every sequent $\Gamma \vdash P$ in the deduction tree, and its corresponding world w built from the sequent in the counter model. We say $\Gamma \vdash P$ and w *correspond to* each other.

Example 3.5 Let's try another formula. For easy reading, we write X, Y, Z instead of $P(X), Q(Y), R(Z)$.

$$\forall X(((\forall Y \forall Z(((Y \to X) \to Z) \to (Y \to Z) \to Z)) \to X) \to X)$$

Let $C(X) = (\forall Y \forall Z(((Y \to X) \to Z) \to (Y \to Z) \to Z)) \to X$

$$
\begin{array}{c}
\overline{C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z \vdash Z}\ \mathit{loop} \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ} \vdash ((Y \to X) \to Z) \to (Y \to Z) \to Z \quad R\to \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z, Y \vdash \forall Y \forall Z(((Y \to X) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z, Y \vdash X \quad L\to \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z \vdash Y \to X \quad R\to \qquad \overline{C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z \vdash Y}\ \mathit{stop} \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z \vdash Z \quad L\to \\
C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ} \vdash ((Y \to X) \to Z) \to (Y \to Z) \to Z \quad R\to \\
C(X), (Y \to X) \to Z, Y \to Z, Y \vdash \forall Y \forall Z(((Y \to X) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(X), (Y \to X) \to Z, Y \to Z, Y \vdash X \quad L\to \\
C(X), (Y \to X) \to Z, Y \to Z \vdash Y \to X \quad R\to \qquad \overline{C(X), (Y \to X) \to Z, Y \to Z \vdash Y}\ \mathit{stop} \\
C(X), (Y \to X) \to Z, Y \to Z \vdash Z \quad L\to \\
C(X) \vdash ((Y \to X) \to Z) \to (Y \to Z) \to Z \quad R\to \\
C(X) \vdash \forall Y \forall Z(((Y \to X) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(X) \vdash X \quad L\to \\
\vdash C(X) \to X \quad R\to \\
\vdash \forall X(C(X) \to X) \quad R\forall
\end{array}
$$

and the c-substitute of the deduction tree is

$$
\begin{array}{c}
\overline{C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e, d, (f \to a) \to g, f \to g \vdash g} \\
C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e, d \vdash ((f \to a) \to g) \to (f \to g) \to g \quad R\to \\
C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e, d \vdash \forall Y \forall Z(((Y \to a) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, b \to e, d \vdash a \quad L\to \\
C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e \vdash d \to a \quad R\to \qquad \overline{C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e \vdash d} \\
C(a), (b \to a) \to c, b \to c, b, (d \to a) \to e, d \to e \vdash e \quad L\to \\
C(a), (b \to a) \to c, b \to c, b \vdash ((d \to a) \to e) \to (d \to e) \to e \quad R\to \\
C(a), (b \to a) \to c, b \to c, b \vdash \forall Y \forall Z(((Y \to a) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(a), (b \to a) \to c, b \to c, b \vdash a \quad L\to \\
C(a), (b \to a) \to c, b \to c \vdash b \to a \quad R\to \qquad \overline{C(a), (b \to a) \to c, b \to c \vdash b} \\
C(a), (b \to a) \to c, b \to c \vdash c \quad L\to \\
C(a) \vdash ((b \to a) \to c) \to (b \to c) \to c \quad R\to \\
C(a) \vdash \forall Y \forall Z(((Y \to a) \to Z) \to (Y \to Z) \to Z) \quad R\forall \\
C(a) \vdash a \quad L\to \\
\vdash C(a) \to a \quad R\to \\
\vdash \forall X(C(X) \to X) \quad R\forall
\end{array}
$$

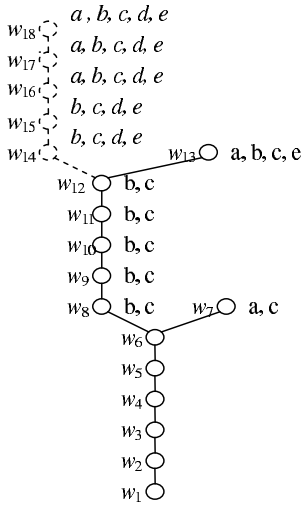

Fig. 2.

The formula was shown to be unprovable in [2]. Here we construct a counter model for it. First, using the algorithm *Search* on the sequent $\vdash \forall X(C(X) \to X)$, we get a *fail tree* which contains two stops and one loop. Next, we make *c-substitute* on every node in the *fail tree* to assign the variables with constants. Then we construct a Kripke model using our method from the leaves to the root. Notice that there exists a loop, and the sequents which *correspond to* w_{18} and w_{12} are both $C(X), [(Y \to X) \to Z, Y \to Z, Y]_{YZ}, (Y \to X) \to Z, Y \to Z \vdash Z$. So we should identify all the notes between w_{18} and w_{12}. Hence the model in real line on the left is our final counter model which is constructed by the worlds $w_1, \ldots, w_{13}$.

4 Correctness of the counter model

In this section, we prove our construction of counter model is correct. That is to say, for a given closed formula E, if E is unprovable then we get model M by the algorithm, and $M \not\models E$.

Remark 4.1 If $Search(\Gamma \vdash A; \xi)$ is not a proof but a deduction tree, then no sequent $\Delta \vdash B$ in the tree is provable.

Lemma 4.2 *Let* $t = Search(\Pi \vdash P; \xi)$*,* w *be a node in the Kripke model* t^m *and* $\Gamma \vdash A$ *be a sequent in* t *which corresponds to* w*, and* $\Sigma \vdash A'$ *is the* c-substitute *of it. Then the following is hold.*
(1) $w \models M$ *for all* $M \in \Sigma$*.*
(2) $w \not\models A'$*.*

Proof. We consider the lowest occurrence of $\Gamma \vdash A$ in t^m.
If w is the leaf of the model labeled with "stop".
Then by the definition, we know A' is atomic, and $A' \notin \Sigma$. Then it is easy to prove (1)and(2) by the definition of the leaves.
Else, let t^* be the tree with w as root, and suppose the result has be proved in all

the nodes of t^* except w.

1. If A is atomic.

1.1. From the definition of t^m we can easy know $w \not\models A'$.

1.2. Next, let's proof $w \models M$

by the definition of the counter model, we know that w has a series of subnodes $w_1, \ldots, w_m$, each w_i *corresponds to* a sequent $\Gamma^i \vdash B_i$,$i \in \{1, \ldots, m\}$

$\Gamma^i = ([\ldots[[\Gamma^i_1]_{V^i_1}, \Gamma^i_2]_{V^i_2}, \ldots, \Gamma^i_{s_i-1}]_{V^i_{s_i-1}}, \Gamma^i_{s_i}, B^i_1 \to \ldots \to B^i_{n_i} \to A) \downarrow$

and Γ can be written as

$\Gamma = \Gamma^i_1, [\Gamma^i_2, [\ldots \Gamma^i_{s_i-1}, [\Gamma^i_{s_i}, B^i_1 \to \ldots \to B^i_{n_i} \to A]_{V^i_{s_i-1}} \ldots]_{V^i_2}]_{V^i_1}$

$B_i \in \{B^i_1, B^i_2, \ldots, B^i_{n_i}\}$

We note the *c-substitute* of $\Gamma^i \vdash B_i$ as $\Sigma^i \vdash B'_i$. By HI we know $w_i \not\models B'_i$, and $w_i \models X$ for all $X \in \Sigma^i$

Since $M \in \Sigma^i$, $w_i \models M$.

1.2.1. If M is atomic.

If there is a sequent $\Gamma_0 \vdash A_0$ whose *c-substitute* $\Sigma_0 \vdash M$ occurs in t^*. Then we have $M \in \Sigma_0$, and this sequent is provable, a contradict. So t^* does not contain a sequent with *c-substitute* $\Sigma_0 \vdash M$, hence $w \models M$ by definition of t^m

1.2.2. If M is not atomic.

Then we have $M = C_1 \to \ldots \to C_n \to N$. N is atomic. We prove $w \models M$ by showing $u \models N$ assuming $u \models C_1, \ldots, u \models C_n$ for an arbitrary $u \geq w$.

1.2.2.1. If $u > w$.

We can find that $u \geq w_i$, $i \in \{1, \ldots, m\}$. Sincne $w_i \models M$, we can get the result simply by definition.

1.2.2.2. If $u = w$.

1.2.2.2.1. t^* does not contain a sequent whose *c-substitute* has the form $\Sigma_0 \vdash N$.

Then by the definition of t^m, $w \models N$, so we have $w \models M = C_1 \to \ldots \to C_n \to N$.

1.2.2.2.2. If t^* contains a sequent $\Gamma_0 \vdash A_0$ whose c-substitute is $\Sigma_0 \vdash N$.

Consider a node v in model t^* that corresponds to $\Gamma_0 \vdash A_0$, $v \geq w$.

1.2.2.2.2.1. If $v > w$

We can know $\exists i \in \{1, \ldots, m\}$, st. $v \geq w_i$. By HI, for v we have $v \not\models N$. Suppose $w \models C_1, \ldots, w \models C_n$, then we have $w_i \models C_1, \ldots, w_i \models C_n$. Since $w_i \models M$, $w_i \models N$ comes, so we get $v \models N$. A contradiction. Thus it does not happen that $w \models C_1, \ldots, w \models C_n$, so $w \models M$.

1.2.2.2.2.2. If $v = w$

Then $N = A'$, $M = C_1 \to \ldots \to C_n \to A'$ is a *c-substitute* of $B^i_1 \to \ldots \to B^i_{n_i} \to A$ for some $i \in \{1, \ldots, m\}$, C_j is a *c-substitute* of B^i_j, j=1,...,n, and $n_i = n$.

Suppose $w \models C_1, \ldots, w \models C_n$, thus $w_i \models C_1, \ldots, w_i \models C_n$. But we know $w_i \not\models B'_i$ by HI, and by the definition of *c-substitute*, we have $B_i \in \{C_1, \ldots, C_n\}$. A contradiction. Then it does not happen that $w \models C_1, \ldots, w \models C_n$, so $w \models M$.

2. If $A = \forall x B$. So the sequent in t is $\Gamma \vdash \forall x B$, and A' is of form $\forall x B'$. It is obtained by using the rule on the sequent $[\Gamma]_V \downarrow \vdash B$ which corresponds to w'. By definition, its *c-substitute* is $\Sigma \vdash B''$, $B'' = B'(s/x)$ and s is a fresh constant not in $\Sigma \vdash B'$. Also we have w' is a child node of w. By definition we have $w' > w$. So we get $w' \models M$ for every $M \in \Sigma$, $w' \not\models B'$ by HI.

2.1. Assuming $w \models A'$, or equally $w \models \forall x B'$. By definition we have $\forall u \geq w, \forall a \in D(u), u \models B(a/x)$. But we have $w' > w$, $w' \not\models B''$. A contradiction. So we have $w \not\models A'$.
2.2. Next we proof $\forall M \in \Sigma, w \models M$.
Firstly, by HI $w' \models M$.
2.2.1. If M is atomic $M \in Support(w')$, by the definition $M \in Support(w)$, so $w \models M$.
2.2.2. If M is not atomic. Then we have $M = C_1 \to \ldots \to C_n \to N$. N is atomic. We proof $w \models M$ by showing $u \models N$ assuming $u \models C_1, \ldots, u \models C_n$ for an arbitrary $u \geq w$.
2.2.2.1. If $u > w$.
We can find that $u \geq w'$. Since $w' \models M$, we can get the result simply by definition.
2.2.2.2. If $u = w$.
2.2.2.2.1. If t^* does not contain a sequent whose *c-substitute* is of the form $\Sigma_0 \vdash N$. Then by the definition of t^m, $w \models N$, so we have $w \models M = C_1 \to \ldots \to C_n \to N$.
2.2.2.2.2. If t^* contains a sequent $\Gamma_0 \vdash A_0$ whose *c-substitute* is of the form $\Sigma_0 \vdash N$.
Consider a node v in model t^m that corresponds to $\Gamma_0 \vdash A_0$, Obviously $v \neq w$, $v \geq w'$. By HI ,$v \not\models N$. Suppose $w \models C_1, \ldots, w \models C_n$, then we have $w' \models C_1, \ldots, w' \models C_n$, since $w' \models M$, we have $w' \models N$, so we get $v \models N$. A contradiction. Thus it does not happen that $w \models C_1, \ldots, w \models C_n$, so $w \models M$.
3. If $A = B \to C$. So the sequent in t is $\Gamma \vdash B \to C$, and A' is of the form $B' \to C'$. It is obtained by using the rule on the sequent $(\Gamma, B) \downarrow \vdash C$ which corresponds to a node w'. By definition its *c-substitute* is $\Sigma, B' \vdash C'$. Also we have w' is a child node of w, $w' > w$. So by HI, $w' \models M$ for every $M \in \Sigma \cup \{B'\}$, $w' \not\models C'$.
3.1. Assuming $w \models A'$, or equally $w \models B' \to C'$. By definition we have $\forall u \geq w$, if $u \models B$ then $u \models C$. But we have $w' > w$, $w' \models B'$,$w' \not\models C'$. A contradiction. So we have $w \not\models A'$.
3.2. Next we proof $\forall M \in \Sigma, w \models M$.
Firstly, by HI $w' \models M$.
3.2.1. If M is atomic $M \in Support(w')$, by the definition, $M \in Support(w)$, so $w \models M$.
3.2.2. If M is not atomic. Then we have $M = C_1 \to \ldots \to C_n \to N$. N is atomic. We proof $w \models M$ by showing $u \models N$ assuming $u \models C_1, \ldots, u \models C_n$ for an arbitrary $u \geq w$.
3.2.2.1. If $u > w$.
We can find that $u \geq w'$. Since $w' \models M$, we can get the result simply by definition.
3.2.2.2. If $u = w$.
3.2.2.2.1. If t^* does not contain a sequent whose *c-substitute* is of the form $\Sigma_0 \vdash N$. Then by the definition of t^m, $w \models N$, so we have $w \models M = C_1 \to \ldots \to C_n \to N$.
3.2.2.2.2. If t^* contains a sequent $\Gamma_0 \vdash A_0$ whose *c-substitute* is of the form $\Sigma_0 \vdash N$.
Consider a node v in model t^m that corresponds to $\Gamma_0 \vdash A_0$, Obviously $v \neq w$, $v \geq w'$. By HI ,$v \not\models N$. Suppose $w \models C_1, \ldots, w \models C_n$, then we have $w' \models C_1, \ldots, w' \models C_n$, since $w' \models M$, we have $w' \models N$, so we get $v \models N$. A contradiction.

Thus it does not happen that $w \models C_1, \ldots, w \models C_n$, so $w \models M$. □

Proposition 4.3 *Given a closed positive formula E. If E is unprovable, and $t = Search(\vdash E, nil)$, $M = t^m$ then $M \not\models E$.*

Proof. By definition, the *c-substitute* of $\vdash E$ is $\vdash E$, applying Lemma 4.2 to the root of model M and sequent $\vdash E$, the theorem follows. □

5 Soundness and completeness

In this section we prove, for a given closed positive formula E in minimal predicate logic, E is provable in system LJB if and only if E is valid.

Definition 5.1 We define a function $Formula$, it assigns a set of formulas to every LJB context Γ. $Formula(\Gamma)$ is a set of all the formulas in Γ' which is obtained from Γ by erasing all the brackets.

Theorem 5.2 (soundness) $\vdash E \Rightarrow \models E$

Proof. We prove, more generally for a given LJB context Γ, if $\Gamma \vdash E$, then for every Kripke model $< W, R, D, V >$ every $w \in W$, and every substitution $\sigma = [x_1/d_1, \ldots, x_n/d_n]$ of objects for free variables in $Formula(\Gamma)$ and E, if every $A \in Formula(\Gamma)$, $w \models A\sigma$, then $w \models E\sigma$.
Let's prove inductively by the reduction of rules.
1. If E is atomic and E is in Γ . The result follows trivially.
2. If the last rule is $L \rightarrow$.
Since $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow E \in Formula(\Gamma)$, we have $w \models (A_1 \rightarrow \ldots \rightarrow A_n \rightarrow E)\sigma$, that means for all $w' \geq w$, if $w' \models A_1\sigma, \ldots, w' \models A_n\sigma$, then $w' \models E\sigma$. By monotonicity, every $A \in Formula(\Gamma)$, $w' \models A\sigma$, we can get for every $A \in Formula(\Gamma')$, $w' \models A\sigma$. Then $w' \models A_1\sigma, \ldots, w' \models A_n\sigma$ by HI, so we get $w' \models E\sigma$. Hence we get $w \models E\sigma$.
3. If the last rule is $R\forall$.
$E = \forall x M$, we extent σ to σ', $\sigma' = \sigma[x/d]$, d is a fresh constant. Equally we want to prove for every $w' \geq w$, $w' \models M\sigma'$. By monotonicity, every $A \in Formula(\Gamma)$, $w' \models A\sigma$, we can get for every $A \in Formula([\Gamma]_V \downarrow)$, $w' \models A\sigma'$. Then $w' \models E\sigma'$ by HI. Hence we get $w \models E\sigma$.
4. If the last rule is $R \rightarrow$.
$E = M \rightarrow N$, equally we try to prove if for every $w' \geq w$, $w' \models M\sigma$, then $w' \models N\sigma$. By monotonicity, every $A \in Formula(\Gamma)$, $w' \models A\sigma$, and since $w' \models M\sigma$, we can get for every $A \in Formula((\Gamma, M) \downarrow)$, $w' \models A\sigma$. Then $w' \models N\sigma$ by HI. Hence we get $w \models E\sigma$. □

Theorem 5.3 (completeness) $\models E \Rightarrow \vdash E$.

Proof. For $\models E$, suppose $\nvdash E$, then by Proposition 4.3, there exists an model M , $M \not\models E$, a contradiction. So the theorem follows. □

Acknowledgement

The authors thank Professor Jiang Ying a lot for many helpful advices for the subject, and thank the anonymous referees whose suggestions have helped improving the paper a lot. Also appreciate all who have offered their help.

References

[1] Dalen, D. v., "Logic and Structure(3rd edition)" Springer,1997.

[2] Dowek, G. and Y. Jiang, *Eigenvariables, bracketing and the decidability of positive minimal predicate logic*, Theoretical Computer Science **360**(2006), pp. 193-208.

[3] Dyckhoff, R., *Contraction-free sequent calculi for intuitionistic logic*, Journal of Symbolic Logic **57**(1995),pp.795-807.

[4] Galmiche, D. and D. J. Pym, *Proof-search in type-theoretic languages: an introduction*. Theoretical Computer Science **232**(2000), pp. 5-53.

[5] Hirokawa, S. and D. Nagano, *Long Normal Form Proof Search and Counter-Model Generation*, Electronic Notes Theorem Computer Science **37**(2000).

[6] Pinto, L., R. Dyckoff, *Loop-free construction of counter-models for intuitionistic propostional logic*, in Behara and al. editors, Symposia Gaussian, 1995, pp. 225-232.

[7] Mints, G., "A Short Introduction to Intuitionistic Logic", Kluwer academic publishers, 2002.

[8] Stoughton, A., *Porgi:a Proof-Or-Refutation Generator for Intuitionistic propositional logic*, in D. Galmiche, editor, CADE Workshop on Proof-search in Type-theoretic Languages, 1996, pp. 109-116