Institute for Logic, Language and Computation

THE RELATIONAL KNOWLEDGE-BASE INTERPRETATION AND FEASIBLE THEOREM PROVING FOR INTUITIONISTIC PROPOSITIONAL LOGIC

Max I. Kanovich

ILLC Prepublication Series
for Mathematical Logic and Foundations ML-93-21

University of Amsterdam

Institute for Logic, Language and Computation
Plantage Muidergracht 24
1018TV Amsterdam
Telephone 020-525.6051, Fax: 020-525.5101

THE RELATIONAL KNOWLEDGE-BASE INTERPRETATION AND FEASIBLE THEOREM PROVING FOR INTUITIONISTIC PROPOSITIONAL LOGIC

Max I. Kanovich
Russian Humanities State University
Moscow, Russia

ILLC Prepublications
for Mathematical Logic and Foundations
ISSN 0928-3315
Coordinating editor: Dick de Jongh

Research partly Supported by the
Netherlands Organization for Scientific Research (NWO)

received December 1993

The Relational Knowledge-Base Interpretation and Feasible Theorem Proving for Intuitionistic Propositional Logic

Max I. Kanovich
Russian Humanities State University, Moscow, Russia

December 15, 1993

Abstract

The decision problem for **Intuitionistic Propositional Logic** is considered:

(i) A computational semantics is introduced for relational knowledge bases. Our semantics naturally arises from practical experience of databases and knowledge bases.
It is stated that the corresponding logic coincides **exactly** with the intuitionistic one.

(ii) Our methods of proof of the general theorems turn out to be very useful for designing new efficient algorithms.
In particular, on the basis of **a specific Calculus of Tasks** related to this computational interpretation, an efficient prove-or-disprove algorithm is designed with the following properties:

- For an arbitrary intuitionistic propositional formula, the algorithm runs in linear deterministic space,
- For every **reasonable** formula, the algorithm runs in **reasonable** time, despite of the fact that in theory it has an 'exponential' uniform lower bound.

Note that in view of the PSPACE-completeness of **Intuitionistic Propositional Logic** an exponential execution time can be expected in the worst case. But such cases only arise for very unnatural formulas, i.e., for formulas that even in their best solutions need maximal cross-linking of all their possible subtasks.

Contents

1 Introduction

In this paper we present the detailed proof for the following results on the complexity and semantics of **Intuitionistic Propositional Logic** that have been obtained in [Kanovich87], and delivered also in [Kanovich90, Kanovich91]:

(1) **Intuitionistic Propositional Logic** is complete with respect to a knowledge-base semantics.

(2) **Intuitionistic Propositional Logic** is recognizable in linear deterministic space.

(3) **Intuitionistic Propositional Logic** is recognizable in quasi-polynomial time: the degree of the polynomial is determined by the *degree of naturality* of formulas under consideration.

2 What we used to have before

Let us observe some problems that have arisen in relation with **Intuitionistic Propositional Logic** .

2.1 Intuitionistic Propositional Logic as a Logic of Tasks

Starting with **A.Kolmogorov and A.Heyting** there were many attempts to interpret **Intuitionistic Propositional Logic** as a logic of tasks. **S.Kleene and Ju.Medvedev** were probably the first who proposed an explicit formalization for **Intuitionistic Propositional Logic** . Unfortunately, all known natural formalizations have led to logics that are essentially stronger than **Intuitionistic Propositional Logic** . Moreover, for many of them, e.g. for the propositional logic of Kleene's realizability, nobody knows whether these logics are decidable or not.

When I started this study in relation with the problem of so-called program synthesis, I (like the most mathematical logicians) was convinced that the corresponding logic would be stronger than **Intuitionistic Propositional Logic** and, probably, undecidable.[1] It should be noted that most troubles are originating from **implication and disjunction.**

2.2 Upper and Lower Bounds for Complexity of Intuitionistic Propositional Logic

'Exponential time.' It is well-known that **Intuitionistic Propositional Logic** is PSPACE-complete [Statman79]. Moreover, we have **a uniform 'exponential' lower bound,** i.e., almost all formulas may require exponential time to be recognized.

Space of the 4th degree. It follows from results of [Ladner77] that **Intuitionistic Propositional Logic** can be solved in space

$$O(L^4),$$

where L is the size of a formula.

As for the decision complexity, most troubles are originating also from **implication and, especially, disjunction.**

3 What we have got now

Let us present our results related to **Intuitionistic Propositional Logic**
.

3.1 The precise computational interpretation

In contrast to what I had expected, I failed to construct a 'computational task' that requires super-intuitionistic rules.

Moreover, to my surprise, I have been able to prove that **Intuitionistic Propositional Logic** is complete under a computational interpretation related to relational databases.

Justifying this statement, a specific **Calculus of Tasks** has been invented along with the corresponding Completeness Theorem.

[1] See comments of G.Mints [Mints83] with respect to the system of automatical program synthesis PRIZ.

3.2 Efficient decision algorithms

As a matter of fact, just the same Completeness Theorem yields very practical consequences for running time and space:

Linear deterministic space. We can recognize **Intuitionistic Propositional Logic** in linear deterministic space

$$O(L).$$

It should be pointed out that such a space is sufficient for **the full set** of intuitionistic connectives.

Quasipolynomial time. We have established **a non-uniform upper bound** for running time:

For a given propositional formula f, we can recognize whether it belongs to **Intuitionistic Propositional Logic** or not, in running time, approximately,

$$L^{r+1}$$

where r is the measure of 'unnaturalness' of formula f:
even in its best solutions f needs cross-linking of, at least, r 'subtasks'.

4 How we got it

Now, I am going to present the main ideas that were used:

- A formula is considered as **an entity, or quantity.** The domain of such an entity may be infinite.
- An entity that represents an implication may be considered as an entity of 'functional' type, a possible value of such a 'functional entity' **should be a program.**
- Possible values of all the entities are collected into **a database,** relations between the entities are perceived as **constraints, or dependencies,** for it.
- **Justifying** a formula means that the corresponding dependency **is satisfied on all 'admissible' databases.**

The following notational conventions are followed throughout this paper:

(I) By letters A, B, C, F, and G
(possibly indexed)
we denote positive literals, or names of 'entities'.

(II) By letters U, V, W, X, Y, and Z
(possibly indexed)
we will denote conjunctions of a number of positive literals.

(III) The **'empty' conjunction** is denoted by $\emptyset$.

4.1 From Formulas to Tasks

Each propositional formula can easily be rewritten in the following form:

$$\Gamma \vdash Z ,$$

where Γ is a multiset consisting of formulas of one of the following three **basic forms:**

(a) $(X \to Y)$,

(b) $((U \to V) \to Y)$,

(c) $(X \to (Y_1 \vee Y_2))$.

Such a **Task Sequent** $\Gamma \vdash Z$ is perceived as a computational task:

Task: Given all laws and dependencies from Γ, compute Z.

Example 4.1
Introducing new names F and G, the propositional formula

$$f_0 : \ ((A\&B \to C) \to (A \to (B \to C)))$$

can be represented by the following task sequent

$$\Gamma_0 \vdash G$$

where Γ_0 consists of five formulas:

$$((B \to C) \to F), (F\&B \to C), ((A \to F) \to G), (G\&A \to F), (A\&B \to C).$$

Here

(I) The following couple of formulas

$$((B \to C) \to F), \ (F \& B \to C)$$

indicates that F is the 'name' of the implication

$$(B \to C).$$

(II) This couple of formulas

$$((A \to F) \to G), \ (G \& A \to F)$$

indicates that G is the 'name' of the implication

$$(A \to F).$$

4.2 A formula as a Knowledge base

The left side Γ of a Task Sequent $\Gamma \vdash Z$ contains all informations that can be used for computing Z and, hence, represents, so to say, **a knowledge base,** i.e. the collection of all laws and dependencies related to our problem.

4.3 The formal definition of a Knowledge base

Now, we give a formal definition of a **relational knowledge base:**

Definition 4.1 A relational knowledge base is a tuple

$$KB = (Names, Functs, Doms, Deps)$$

where

(1)

$$Names = (A_1, \ A_2, \ A_3, \ \ldots, \ A_n, \ \ldots)$$

is a recursively enumerable sequence of literals or, in other words, names of "entities" (or "attributes").

(2) The set $Names$ is divided into two parts: some entities are declared as "functional entities".

Functs is a recursively enumerable set of names of all "functional entities" together with their finite types:

Definition 4.2 **The type of a functional entity** F is an expression of the form

$$(U \to V).$$ [2]

(3) *Doms* is a recursively enumerable sequence of domains of entities from *Names*:
the domain of entity A is denoted by $Dom(A)$. [3]

For a given X,
$Dom(X)$ is the Cartesian product of domains of all entities from X.

The crucial point of the definition is that, for functional entities, we require that in order to specify some concrete value of a functional entity it is not sufficient to give its set-theoretical description; instead, it is necessary to present a program calculating this function.

Therefore, the domain of a functional entity F of the type $(U \to V)$ is defined **to be the set of all programs** mapping $Dom(U)$ into $Dom(V)$.

(4) *Deps* is a recursively enumerable set of laws, constraints, dependencies etc. connected with our problem area.

In this paper *Deps* consisits of functional, operator and variant dependencies (described below in Section 4.4).

Example 4.2 *(continuing Example 4.1)*
With Γ_0 we can associate the following knowledge base

$$KB_0 = (Names_0, Functs_0, Doms_0, Deps_0)$$

where

- $Names_0 = (A, B, C, F, G)$
- $Functs_0$ consists of two functional entities:
  (1) F of the type $(B \to C)$, and
  (2) G of the type $(A \to F)$.

[2]Both U and V must be non-empty. U is called **the input-of-functional.**
[3]An **empty entity** B such that $Dom(B)$ is empty may be considered.

- $Deps_0$ is the singleton:

$$(A\&B \to C)$$

With a given knowledge base KB we can associate the multiset of basic formulas $Axioms(KB)$ as follows:

Definition 4.3 A functional entity F of the type $(U \to V)$ is represented by the set $Axiom_F$ consisting of the following two formulas:

$$((U \to V) \to F), \quad (F\&U \to V).$$

Definition 4.4 By $Axioms(KB)$ we denote union of

(1) the set $Deps$ from KB and

(2) union of the sets $Axiom_F$, for all functional entities F from KB.

Example 4.3 *(continuing Example 4.2)*
For this KB_0, we have

$$Axioms(KB_0) = \Gamma_0.$$

4.4 A basic formula as a constraint, or a dependency

Each basic formula is considered as a representation of some constraint:

(a) An implication

$$(X \to Y)$$

should be perceived as the **functional dependency:**

> "There is a **computable** function (a functional of higher type) from $Dom(X)$ into $Dom(Y)$". [4]

(b) An embedded implication

$$((U \to V) \to Y)$$

is treated as the **operator dependency**

$$(F \to Y)$$

where F is a functional entity of type $(U \to V)$.

[4] It should be noted that names of functional entities may be contained in X and Y.

(c) An implication along with disjunction

$$(X \to (Y_1 \vee Y_2))$$

should be taken as the **variant dependency:**

> "For some values of X, the values of Y_1 can be calculated, and,
> for the rest of the values of X, the values of Y_2 can be computed". [5]

4.5 What does it mean 'TO BE FALSE' ?

We may use an entity B such that $Dom(B)$ is empty, **to represent falsity.**

Following this idea, **negation of** U is represented with the help of a functional entity of the type

$$(U \to B)$$

where $Dom(B)$ is empty.

5 What does it mean 'BEING SOLVABLE' ?

5.1 Relational databases

As models for a knowledge base we consider relational databases:

First, define the notion of a "possible state". We assume that there is a symbol UNDEF that represents undefinedness.

Definition 5.1

- A recursively enumerable sequence

$$s = (a_1,\ a_2,\ a_3,\ \ldots,\ a_n,\ \ldots)$$

  where for every n, a_n is from $Dom(A_n)$ or is undefined (equals UNDEF), is called the **state** of an object.

- Every a_n is denoted also by $A_n(s)$ and is treated as *"the value of the entity A_n in the state s".*

[5] Y_1 and Y_2 are called **alternatives.**

- *For* $X = B_1 \& B_2 \& \cdots \& B_m$, *we will write*

$$X(s) = (B_1(s), B_2(s), \dots, B_m(s)),$$

if some $B_j(s)$ *is equal to* UNDEF *we shall take* $X(s)$ *to be undefined:*

$$X(s) = \text{UNDEF} \ .$$

Definition 5.2
An arbitrary recursively enumerable set of states is called **(an instance of) a database.**

Example 5.1 *(continuing Example 4.3)*
We may consider the following database T from Table 1.

A	B	C	F	G
1	1	2	p_2	q_2
1	2	3	p_3	q_3
2	1	3	p_3	q_3

(a)

A	B	C	F	G
1	1	2	p_2	q_2
1	2	3	p_3	q_3

(b)

where p_2 and p_3 are programs mapping any input into 2 and 3, respectively,
q_2 and q_3 are programs mapping any input into p_2 and p_3, respectively.

Table 1: (a) Database T. (b) Database $T/_{A=1}$.

5.2 Four possible versions for 'BEING SOLVABLE'

We are interested in instances of databases that are **consistent with all the laws** from a given relational knowledge base.
In our definitions we adhere the following

Principle of conservation: WHEN THE VALUES OF ENTITIES ARE KEPT FIXED, ALL THE LAWS MUST BE PRESERVED.

For functional and variant dependencies, **this principle holds automatically.**

As for the operator dependency

$$((U \to V) \to Y),$$

it can be elaborated as follows:[6]

> For a given database T, when we fix values for the entities from W, say by setting:
>
> $$W = w,$$
>
> we thus truncate our database:
>
> **Definition 5.3** Henceforth, by $T/_{W=w}$ we will denote this truncated database, i.e. the set of all states s from T such that
>
> $$W(s) = w.$$
>
> Suppose that on the new truncated database the dependency between U and V becomes functional and there is a program p such that
>
> $$V = p(U)$$
>
> on $T/_{W=w}$.
>
> According to the operator dependency $((U \to V) \to Y)$, having such a program p, we can compute Y, and, in other words, the functional dependency $(\emptyset \to Y)$ **should be satisfied** on $T/_{W=w}$.

Definition 5.4

For basic dependencies:

(a) We say that **a functional dependency**

$$(X \to Y)$$

is satisfied on an instance T if there exists a program g mapping $Dom(X)$ into $Dom(Y)$ such that, for every state s from T, if $X(s)$ is defined then $Y(s)$ is defined and

$$Y(s) = g(X(s)).$$

[6] cf. also Corollary 5.1

(b) **An operator dependency**

$$((U \to V) \to Y)$$

is satisfied on an instance T if, for arbitrary W and a tuple w from $Dom(W)$, if the functional dependency $(U \to V)$ is satisfied on the truncated database $T/_{W=w}$ then the functional dependency $(\emptyset \to Y)$ is satisfied on $T/_{W=w}$. [7]

(c) **A variant dependency**

$$(X \to (Y_1 \vee Y_2))$$

is said to be satisfied on an instance T if there exists a program q mapping $Dom(X)$ into the set {1,2}, a program g_1 mapping $Dom(X)$ into $Dom(Y_1)$, and a program g_2 mapping $Dom(X)$ into $Dom(Y_2)$ such that, for every state s from T, if $X(s)$ is defined then

1. if $q(X(s)) = 1$ then $Y_1(s)$ is defined and $Y_1(s) = g_1(X(s))$,
2. if $q(X(s)) = 2$ then $Y_2(s)$ is defined and $Y_2(s) = g_2(X(s))$.

(d) We say that an instance T is **in full accordance with a functional entity** F of the type $(U \to V)$ if

(I) for every state s from T, if both $U(s)$ and $F(s)$ are defined then $V(s)$ is defined and[8]

$$V(s) = F(s)(U(s)),$$

(II) the operator dependency $((U \to V) \to F)$ is satisfied on T.

---

[7]If we want **to ignore this principle of conservation** and change this item by requiring that W **instead of being arbitrary be empty** , namely,
if we replace this item with the following item:

(b') For **an operator dependency**

$$((U \to V) \to Y),$$

we say that it is satisfied on an instance T in the following case:
if the functional dependency $(U \to V)$ is satisfied on T then the functional dependency $(\emptyset \to Y)$ is also satisfied on T.

then we get a characterization related to modal logics S4 and S5. See also Example 5.2.

[8]Let us recall that $F(s)$ is a program of the type $(U \to V)$

Corollary 5.1 *Let a database T be in full accordance with a functional entity F of the type $(U \to V)$.*

For a given W, let

$$(W\&U \to V)$$

be satisfied on a database T.

Then the functional dependency

$$(W \to F)$$

is satisfied on the whole T. [9]

Proof. Let g be a program computing V from the conjunction $W\&U$ on the whole database T.

For a given w, on the following truncated database $T/_{W=w}$ the dependency between U and V becomes functional, moreover, one can extract a program p from g such that

$$V = p(U)$$

on $T/_{W=w}$.

According to Definition 5.4, the dependency $(\emptyset \to F)$ is satisfied on $T/_{W=w}$, and, having such a program p, we can construct implementation of F on $T/_{W=w}$.

Hence, for each w, the value of F is computed, and the dependency $(W \to F)$ is satisfied on the whole T. ■

Definition 5.5 An instance T is called **consistent with a knowledge base** KB if

(1) all the dependencies from KB are satisfied on T,

(2) T is in full accordance with every functional entity F from KB.

Definition 5.6 (Solvability) For a given class of instances K, we say that a task sequent $Axioms(KB) \vdash Z$ is K-solvable if, for every instance T from K, if T is consistent with KB, then the functional dependency $(\emptyset \to Z)$ is satisfied on T.

May be it will be reasonable to insert some example here.

[9] It should be noted that this item is **directly correlated with Kleene's s-m-n theorem [Rogers67].**

Example 5.2 *(continuing Example 5.1)*

It seems that the database T from Example 5.1 **rejects** our **valid formula** f_0 from Example 4.1 because

(1) the dependency $(A\&B \to C)$ is satisfied on T,

(2) T is in accordance with both functional entities F and G, but

(3) the dependency $(\emptyset \to G)$ is not satisfied on T.

The point is that **there is no full accordance** between T and F because **the principle of conservation is violated.**

E.g., if we truncate T by setting:

$$A = 1,$$

then, on the truncated database $T/_{A=1}$ from Table 1 the relation between B and C becomes functional and, hence, $(\emptyset \to F)$ should be satisfied on $T/_{A=1}$, which is a contradiction.

Theorem 5.1 (Robustness and Completeness) .

Let KB be a finite knowledge base.[10]

For every A , let $Dom(A)$ be infinite or empty.[11]

Let K be

(a) either the class of all databases, or

(b) the class of all finite databases.

Then, for a given task sequent $Axioms(KB) \vdash Z$ the following sentences are equivalent pairwise:

(i) $Axioms(KB) \vdash Z$ is K-solvable,

(ii) on replacing all computable functions with the corresponding set-theoretical functions in the definitions related to the concept of consistent databases, the task $Axioms(KB) \vdash Z$ is K-solvable in this new sense,

(iii) one can construct a program for the task $Axioms(KB) \vdash Z$,

[10] Theorem 5.1 is valid for all finite knowledge bases. As for infinite knowledge bases cf. section 8

[11] This hypothesis is essential!

(iv) $Axioms(KB) \vdash Z$ *can be derived in* **the Calculus of Tasks** *(described below).*

Proof. It follows from Theorem 6.1 and Theorem 6.2. ∎

Theorem 5.1 shows that all reasonable definitions are equivalent and demonstrates that our definition of solvable tasks **is very robust and does not depend on** the particular choice of a level of constructivity.

Corollary 5.2 *Under the hypotheses of Theorem 5.1, a sequent*

$$Axioms(KB),\ (\emptyset \to X) \vdash Z$$

is derivable in **the Calculus of Tasks if and only if** *for every database* T *(from* K*) that is consistent with* KB **(saying nothing about** $(\emptyset \to X)$ **)**, *the functional dependency*

$$(X \to Z)$$

is satisfied on T.

6 The Calculus of Tasks

All theorems are based on **the Calculus of Tasks** that operates with task sequents.

6.1 The language of the Calculus of Tasks

Let us recall that a **task sequent** is of the form

$$\Gamma \vdash Z\ ,$$

where Γ is a multiset consisting of formulas of one of the following three **basic forms:**

(a) $(X \to Y)$,

(b) $((U \to V) \to Y)$,

(c) $(X \to (Y_1 \vee Y_2))$.

6.2 Axioms for the Calculus of Tasks

Definition 6.1 For a multiset of formulas Γ, by $Out(\Gamma)$ we denote the set of all B such that the formula

$$(\emptyset \to B)$$

is contained in Γ.

Definition 6.2 A sequent

$$\Gamma \vdash Z$$

is called **an axiom** if either

(a) Z is contained in $Out(\Gamma)$, or

(b) for some B such that $Dom(B)$ is empty, B is contained in $Out(\Gamma)$.

6.3 Inference Rules for the Calculus of Tasks

Let us give the inference rules for **the Calculus of Tasks** :

Conjunction: $$\frac{\Gamma,\ (\emptyset \to Y),\ (\emptyset \to B) \vdash Z}{\Gamma,\ (\emptyset \to Y\&B) \vdash Z}$$

Composition$_1$: $$\frac{\Gamma,\ (X \to Y),\ (\emptyset \to B) \vdash Z}{\Gamma,\ (X\&B \to Y),\ (\emptyset \to B) \vdash Z}$$

Composition$_2$: $$\frac{\Gamma,\ (X \to (Y_1 \vee Y_2)),\ (\emptyset \to B) \vdash Z}{\Gamma,\ (X\&B \to (Y_1 \vee Y_2)),\ (\emptyset \to B) \vdash Z}$$

Subprogram: $$\frac{\Gamma,\ (\emptyset \to U) \vdash V \qquad \Gamma,\ (\emptyset \to Y) \vdash Z}{\Gamma,\ ((U \to V) \to Y) \vdash Z}$$

Branching: $$\frac{\Gamma,\ (\emptyset \to Y_1) \vdash Z \qquad \Gamma,\ (\emptyset \to Y_2) \vdash Z}{\Gamma,\ (\emptyset \to (Y_1 \vee Y_2)) \vdash Z}$$

Definition 6.3 **A derivation in the Calculus of Tasks** is a finite sequence of task sequents such that each member of it is either an axiom sequent or the result of application of one of the inference rules to preceding sequents.

6.4 Completeness of the Calculus of Tasks

Theorem 6.1 (Soundness) *For a given knowledge base* KB [12], *if a task sequent*

$$Axioms(KB) \vdash Z$$

is derivable in **the Calculus of Tasks** *then this task sequent is solvable in all senses of Theorem 5.1.*

Proof. Follows from that all rules of **the Calculus of Tasks** are correct with respect to each of our semantics. ■

Theorem 6.2 (Completeness) *Let* KB *be a finite knowledge base and, for each entity* A, *let* $Dom(A)$ *be either infinite or empty.*[13]

If a task sequent

$$Axioms(KB) \vdash Z_0$$

is not derivable in **the Calculus of Tasks** , *then we can construct a finite database* T *such that*

- T *is consistent with* KB, *but*
- $(\emptyset \to Z_0)$ *is not satisfied on* T.

Proof. We construct a refutable database in two steps:

(I) First, we make up a Kripke-like skeleton.

(II) Then, we build up the flesh of computational details on it.

Step 1: From the sequent to a Kripke-like skeleton

Definition 6.4 We say that **a sequent** S **is reduced to a sequent** S_1 if either

(Inverting the Conjunction rule) S is of the form

$$\Gamma,\ (\emptyset \to Y\&B) \vdash Z\ ,$$

and S_1 is of the form

$$\Gamma,\ (\emptyset \to Y),\ (\emptyset \to B) \vdash Z\ ,$$

or

[12] KB may be infinite

[13] See remarks in Section 6.7

(Inverting the Composition$_1$ rule) S is of the form

$$\Gamma,\ (X\&B \to Y),\ (\emptyset \to B) \vdash Z\ ,$$

and S_1 is of the form

$$\Gamma,\ (X \to Y),\ (\emptyset \to B) \vdash Z\ ,$$

or

(Inverting the Composition$_2$ rule) S is of the form

$$\Gamma,\ (X\&B \to (Y_1 \vee Y_2)),\ (\emptyset \to B) \vdash Z\ ,$$

and S_1 is of the form

$$\Gamma,\ (X \to (Y_1 \vee Y_2)),\ (\emptyset \to B) \vdash Z\ ,$$

or

(Partial inverting the Subprogram rule) S is of the form

$$\Gamma,\ ((U \to V) \to Y) \vdash Z\ ,$$

and S_1 is of the form

$$\Gamma,\ (\emptyset \to Y) \vdash Z\ ,$$

and **the sequent**

$$\Gamma,\ (\emptyset \to U) \vdash V$$

is derivable in the Calculus of Tasks .

Definition 6.5 A sequent S_1 is said to be **a reduction limit of a sequent** S if

(a) S can be transformed into S_1 by means of a finite sequence of reductions, and

(b) S_1 cannot be reduced to any sequent.

Lemma 6.1 *Let a sequent*

$$\Gamma_1 \vdash Z$$

be a reduction limit of a sequent

$$\Gamma \vdash Z\ .$$

Then

(a) $Out(\Gamma) \subseteq Out(\Gamma_1)$.

(b) *If* $\Gamma_1 \vdash Z$ *is derivable in* **the Calculus of Tasks** *then* $\Gamma \vdash Z$ *is also derivable in* **the Calculus of Tasks** .

Definition 6.6 Now we construct **a tree of limit sequents** as follows:

- The root is claimed to be a reduction limit of the sequent

$$Axioms(KB) \vdash Z_0 \ .$$

- For every vertex v of the form

$$\Gamma, \ ((U \rightarrow V) \rightarrow Y) \vdash Z \ ,$$

  we construct a new son v_1 of v such that v_1 is a reduction limit of the sequent

$$\Gamma, \ (\emptyset \rightarrow U) \vdash V \ .$$

- For every vertex v of the form

$$\Gamma, \ (\emptyset \rightarrow (Y_1 \vee Y_2)) \vdash Z \ ,$$

  we

  (1) select **a non-derivable sequent** S from the following two sequents:

$$\Gamma, \ (\emptyset \rightarrow Y_1) \vdash Z$$

  and

$$\Gamma, \ (\emptyset \rightarrow Y_2) \vdash Z \ ,$$

  (2) then, construct a new son v_1 of v such that v_1 is a reduction limit of S.

Lemma 6.2 *(A) Such a tree is to be finite, and*

(B) Each vertex of it is not derivable in **the Calculus of Tasks** .

Proof. It follows from Lemma 6.1. ■

Definition 6.7 For a vertex v of the form

$$\Gamma \vdash Z \ ,$$

the set $Out(\Gamma)$ will be also denoted by $Out(v)$.

Lemma 6.3 *Let* v_1 *be a descendant of* v. *Then*

$$Out(v) \subseteq Out(v_1).$$

Proof. It follows from Lemma 6.1. ∎

Lemma 6.4 *For every formula* $(X \to Y)$ *from* $Axioms(KB)$ *,*
if $X \subseteq Out(v)$ *then* $Y \subseteq Out(v)$. [14]

Proof. We use that v is a reduction limit. ∎

Lemma 6.5 *Let* v *be of the form*

$$\Gamma \vdash Z_1 \; .$$

For every formula $(X \to (Y_1 \vee Y_2))$ *from* $Axioms(KB)$ *,*
if $X \subseteq Out(v)$ *then one can find a descendant* v_1 *of* v *such that*

(a) v_1 *is of the form*

$$\Gamma_1 \vdash Z_1 \; ,$$

(b) either Y_1 *or* Y_2 *is contained in* $Out(v_1)$.

Proof. Suppose that neither Y_1 nor Y_2 are contained in $Out(v)$.

Then a formula of the form $(X_1 \to (Y_1 \vee Y_2))$ is to be contained in Γ, where X_1 is the result of contractions of X. Since v is a reduction limit, X_1 **must be empty.** According to Definition 6.6, there is a son v_1 of v such that

(a) v_1 is of the form $\Gamma_1 \vdash Z_1$, and

(b) either Y_1 or Y_2 is contained in $Out(v_1)$. ∎

Definition 6.8 We say that a vertex v is **good** if, for every formula

$$(X \to (Y_1 \vee Y_2))$$

from $Axioms(KB)$, if $X \subseteq Out(v)$ then either Y_1 or Y_2 is contained in $Out(v)$.

The following lemma **is vital for variant dependencies.**

---

[14] $X \subseteq Out(v)$ means that each B from X is contained in $Out(v)$.

Lemma 6.6 *For every vertex* v *of the form*

$$\Gamma \vdash Z_1 \ ,$$

one can find **a good descendant** v_1 **of it** *such that* v_1 *is of the form*

$$\Gamma_1 \vdash Z_1 \ .$$

Proof. Taking into account Lemma 6.3, by using Lemma 6.5 iteratively, we can find a required good descendant of v. ■

The tree consisting of all good vertices along with the evaluation function Out represents **a refutable Kripke model:**

Lemma 6.7 *For each vertex* v*:*

(a) *For every formula* $(X \to Y)$ *from* $Axioms(KB)$ *,*
if $X \subseteq Out(v)$ *then* $Y \subseteq Out(v)$*.*

(b) *For every formula* $((U \to V) \to Y)$ *from* $Axioms(KB)$ *,*
if, for all good descendants v_1 *of* v *such that*

$$U \subseteq Out(v_1),$$

$\mathring{V}$ *is contained in* $Out(v_1)$ *then* Y *is contained in* $Out(v)$*.*

(c) *For every formula* $(X \to (Y_1 \vee Y_2))$ *from* $Axioms(KB)$ *,*
if $X \subseteq Out(v)$ *and* v **is good** *then either* Y_1 *or* Y_2 *is contained in* $Out(v)$*.*

Proof.

(a) It follows from Lemma 6.4.

(b) Assume that Y is not contained in $Out(v)$.
Then v is of the form

$$\Gamma, \ ((U \to V) \to Y) \vdash Z \ .$$

Let us consider the son v_1 of it such that v_1 is of the form

$$\Gamma_1, \ (\emptyset \to U) \vdash V \ .$$

By Lemma 6.6, there is a good descendant v_2 of it such that v_2 is of the form

$$\Gamma_2, \ (\emptyset \to U) \vdash V \ .$$

For this good v_2, we have:

- $U \subseteq Out(v_2)$, but
- Lemma 6.2 implies that V is not contained in $Out(v_2)$,

which is a contradiction.

(c) It follows from Definition 6.8. ∎

Step 2: From the Kripke-like skeleton to a database

Now, we will develop a refutable database.

Without loss of generality, we consider the following case:

- For every simple entity A,
  $Dom(A)$ is either the set of all non-negative numbers or empty.
- For every functional entity A,
  $Dom(A)$ is the set of all Gödel numbers of all partial recursive functions.
- For every functional entity F of the type $(U \to V)$,
  the length of both U and V is equal to 1,
- There is no operator dependencies.

Let

$$\varphi_0,\ \varphi_1,\ \varphi_2,\ \ldots,\ \varphi_n,\ \ldots$$

be a Gödel enumeration of all partial recursive functions [Rogers67].

Lemma 6.8 *There exists an increasing primitive recursive function t such that, for every partial recursive function h, one can realize an integer* **(a fixed point)** *b such that, for all i, v and x,*

$$\varphi_{t(b,i,v)}(x) = h(b,i,v,x).$$

Proof. By s-m-n theorem [Rogers67], there is an increasing primitive recursive function t such that, for all z, i, v and x,

$$\varphi_{t(z,i,v)}(x) = \psi_3(z,i,v,x)$$

where ψ_3 is a universal function for all ternary partial recursive functions.

According to the Kleene's Recursion Theorem [Rogers67], for every partial recursive function h, we can find b such that, for all i, v and x,

$$\psi_3(b,i,v,x) = h(b,i,v,x)$$

and, hence,

$$\varphi_{t(b,i,v)}(x) = \psi_3(b,i,v,x) = h(b,i,v,x).$$

■

We introduce an auxiliary function *Eval* by the following:

Definition 6.9

(i) For the root v, we set: [15]

$$Eval(z,i,v) = \begin{cases} t(z,i,v), & \text{if } A_i \in Out(v), \\ \text{UNDEF} & \text{otherwise.} \end{cases}$$

(ii) For a son v_1 of a vertex v, we set:

$$Eval(z,i,v_1) = \begin{cases} t(z,i,v_1), & \text{if } \ v_1 \text{ is good, and } A_i \in Out(v_1), \\ & \qquad \text{and } \ Eval(z,i,v) = \text{UNDEF} \ , \\ Eval(z,i,v), & \text{otherwise.} \end{cases}$$

Definition 6.10 All good vertices and the root will be called **basic vertices.**

Our definitions imply the following

Lemma 6.9 *If $Eval(z,i,v)$ is defined then there is a basic ancestor v_0 of v such that*

$$Eval(z,i,v) = Eval(z,i,v_0) = t(z,i,v_0).$$

Definition 6.11 We construct a partial recursive function h such that, for all z, i, v and x,

(1) $h(z,i,v,0) = v$,

[15] For simplicity, we ignore the difference between integers and words that represent finite sequents

(2) for each functional entity A_i of the type $(A_j \to A_k)$ that $Eval(z,i,v)$ is defined, and for every descendant v_1 of v that $Eval(z,j,v_1)$ is defined, there is a descendant v_2 of v such that

$$Eval(z,j,v_2) = Eval(z,j,v_1)$$

and

$$h(z,i,v,Eval(z,j,v_1)) = Eval(z,k,v_2).$$

Lemma 6.10 *For our function h from Definition 6.11, following Lemma 6.8, we find an integer b such that, for all i, v and x,*

$$\varphi_{t(b,i,v)}(x) = h(b,i,v,x).$$

Let us define **an evaluation function** *eval* as follows:

Definition 6.12

For all i and v:

$$eval(i,v) = Eval(b,i,v)$$

where b is the integer from Lemma 6.10.

Definition 6.13

A state s_v is assigned to every vertex v as follows:

$$s_v = (eval(1,v),\ \ eval(2,v),\ \ eval(3,v),\ \ \ldots,\ \ eval(n,v),\ \ \ldots).$$

Definition 6.14

(i) Let T be the set of all such s_v.

(ii) For a vertex v, by T_v we will denote the set of all states assigned to descendants of v.

Now, we will enumerate **the main peculiarities of our function** *eval***.**

Lemma 6.11 *(a) If $eval(i,v)$ is defined then $A_i \in Out(v)$.*

(b) For a basic vertex v, if $A_i \in Out(v)$ then $eval(i,v)$ is defined.

(c) If both $eval(i,v_1)$ *and* $eval(i,v_2)$ *are defined and*

$$\varphi_{eval(i,v_1)} = \varphi_{eval(i,v_2)}$$

then there is a basic vertex v *such that*

- v *is a common ancestor of* v_1 *and* v_2*, and*
- $eval(i,v_1) = eval(i,v_2) = eval(i,v)$.

(d) If $eval(i,v)$ *is defined and,*
for a descendant v_1 *of* v, $eval(j,v_1)$ *is defined*
then $eval(k,v_1)$ *is to be defined and*

$$\varphi_{eval(i,v)}(eval(j,v_1)) = eval(k,v_1).$$

Proof.

(a) It follows from Definition 6.9.

(b) It also follows from Definition 6.9.

(c) Following Lemma 6.9, let v_3 be a basic ancestor of v_1 such that

$$eval(i,v_1) = eval(i,v_3) = t(b,i,v_3),$$

and let v_4 be a basic ancestor of v_2 such that

$$eval(i,v_2) = eval(i,v_4) = t(b,i,v_4).$$

According to Lemma 6.8 and Definition 6.11,

$$\varphi_{eval(i,v_1)}(0) = \varphi_{t(b,i,v_3)}(0) = h(b,i,v_3,0) = v_3$$

and

$$\varphi_{eval(i,v_2)}(0) = \varphi_{t(b,i,v_4)}(0) = h(b,i,v_4,0) = v_4.$$

Hence,

$$v_3 = v_4.$$

(d) In this item we use the following two lemmas.

Lemma 6.12 *For arbitrary* W *and a tuple* w *from* $Dom(W)$*, there exists a basic vertex* v *such that*

$$T/_{W=w} = T_v.$$

Proof. To use induction on the length of W, it is sufficient to consider a database of the form $T/_{A_i=a}$.

Suppose that, for vertices v_1 and v_2,

$$eval(i, v_1) = eval(i, v_2) = a.$$

Let v be **the most remote** basic ancestor of v_1 such that

$$eval(i, v) = eval(i, v_1) = a$$

and let v_4 be **the most remote** basic ancestor of v_2 such that

$$eval(i, v_4) = eval(i, v_2) = a.$$

Item (c) of Lemma 6.11 implies that

$$v = v_4.$$

Hence, we can conclude that

$$T/_{A_i=a} \subseteq T_v.$$

On the other hand, from Definition 6.9 follows that

$$T_v \subseteq T/_{A_i=a}.$$

■

Lemma 6.13 *For every formula* $(X \to Y)$ *from* $Axioms(KB)$ *, the functional dependency* $(X \to Y)$ *is satisfied on* T.

Proof. For given vertices v_1 and v_2, let $X(s_{v_1})$ be defined and

$$X(s_{v_1}) = X(s_{v_2}) = x.$$

By Lemma 6.12, there is a basic vertex v such that v is a common ancestor of v_1 and v_2, and

$$X(s_v) = x.$$

Since $X \subseteq Out(v)$, Lemma 6.7 implies that $Y \subseteq Out(v)$ and, therefore, $Y(s_v)$ is defined and

$$Y(s_{v_1}) = Y(s_{v_2}) = Y(s_v).$$

■

(d) Going back to point (d) of Lemma 6.11,
let v_0 be a basic ancestor of v such that

$$eval(i, v) = eval(i, v_0) = t(b, i, v_0).$$

Then, according to Definition 6.11, for some descendant v_2 of v_0, we have:

$$eval(j, v_2) = eval(j, v_1)$$

and

$$h(b, i, v_0, eval(j, v_1)) = eval(k, v_2).$$

Since the formula $(A_i \& A_j \to A_k)$ is in $Axioms(KB)$, Lemma 6.13 implies that $eval(k, v_2)$ is defined and

$$eval(k, v_2) = eval(k, v_1).$$

According to Lemma 6.8,

$$\begin{aligned} \varphi_{eval(i,v)}(eval(j, v_1)) &= \varphi_{t(b,i,v_0)}(eval(j, v_1)) \\ &= h(b, i, v_0, eval(j, v_1)) \\ &= eval(k, v_2) \\ &= eval(k, v_1). \end{aligned}$$

■

Lemma 6.14 *For each functional entity* A_i *of the type* $(A_j \to A_k)$ *, the instance* T *is in full accordance with* A_i*.*

Proof. We have to prove the both lines of item (d) in Definition 5.4.

(I) It follows from item (d) of Lemma 6.11.

(II) Let us examine the operator dependency

$$((A_j \to A_k) \to A_i).$$

For given W and w from $Dom(W)$, suppose that the functional dependency $(A_j \to A_k)$ is satisfied on the truncated database $T/_{W=w}$.

By Lemma 6.12, there exists a basic vertex v such that

$$T/_{W=w} \;=\; T_v.$$

Item (b) of Lemma 6.7 shows that $A_i \in Out(v)$. Hence, for every state s from T_v, we have:

$$A_i(s) = A_i(s_v).$$

We can conclude that the functional dependency $(\emptyset \to A_i)$ is satisfied on $T/_{W=w}$. ■

Lemma 6.15 *Let v be a good vertex.*

Every variant dependency $(X \to (Y_1 \vee Y_2))$ from KB is satisfied on the database T_v.

Proof. For a given x from $Dom(X)$, let us examine the instance $T_v/_{X=x}$.

By Lemma 6.12, there exists **a good descendant** v^1 **of** v such that

$$T_v/_{X=x} \ = \ T_{v^1}.$$

Since $X \subseteq Out(v^1)$, then either Y_1 or Y_2 is contained in $Out(v^1)$. Therefore, either $(\emptyset \to Y_1)$ or $(\emptyset \to Y_2)$ is satisfied on T_{v^1}. ■

Lemma 6.16 *For every good vertex v, the instance T_v is consistent with KB.*

Proof. It follows from Lemma 6.13, Lemma 6.15 and Lemma 6.14. ■

Now, we may complete the proof of the Completeness Theorem 6.2.

Taking into account that the root of our tree is of the form

$$\Gamma \vdash Z_0 \ ,$$

with the help of Lemma 6.6, we find **a good descendant** v such that v **is of the form**

$$\Gamma_1 \vdash Z_0 \ .$$

For this good v,

- Lemma 6.16 implies that the instance T_v is consistent with KB, but
- Lemma 6.2 shows that Z_0 is not contained in $Out(v)$ and the functional dependency $(\emptyset \to Z_0)$ is not satisfied on T_v.

And we complete the proof Completeness Theorem 6.2. ■

6.5 The programmer's interpretation of the Calculus of Tasks

Each inference rule can be perceived as a **formalization of a constructive step in a natural reasoning** related to solving tasks:

(Composition) If a knowledge base contains a dependency

$$(\emptyset \to B)$$

(that means B is computed) and a dependency

$$(B \to Y)$$

(that means Y is computed from B) then we can compute Y with the help of a composition and, therefore, the true law

$$(\emptyset \to Y)$$

can be added to the knowledge base.

(Subprogram) If, for a functional entity F of the type $(U \to V)$, we want to add the dependency

$$(\emptyset \to F)$$

("a procedure F is implemented") to the set of laws Γ, then , in advance, we must synthesize a subprogram for this F, in other words solve the **subtask**

$$\Gamma,\ (\emptyset \to U) \vdash V\ .$$

(Branching) If, for a variant dependency

$$(\emptyset \to (Y_1 \vee Y_2))$$

that means either Y_1 or Y_2 is computed, we are able to solve both **subtasks:**

$$\Gamma,\ (\emptyset \to Y_1) \vdash Z$$

and

$$\Gamma,\ (\emptyset \to Y_2) \vdash Z\ ,$$

then we can solve the main task.

Corollary 6.1 *Taking into account what has been said, a program can be extracted* **directly** *from a derivation in* **the Calculus of Tasks** *.*

Corollary 6.2 *This minimal set of rules of reasoning turns out to cover* **all possible rules of reasoning for solving tasks, which could be thought of on the propositional level.**

6.6 The Calculus of Tasks is an information preserving calculus

To speed up our search for derivations, we have used the freedom of choice provided by the following **unexpected corollary.**

Corollary 6.3 *Each of the rules of* **the Calculus of Tasks** *preserves all initial information.*
More specifically,

For the Conjunction rule: *its conclusion is derivable* **if and only if** *its premise is derivable.*

For the Composition$_1$ rule: *its conclusion is derivable* **if and only if** *its premise is derivable.*

For the Composition$_2$ rule: *its conclusion is derivable* **if and only if** *its premise is derivable.*

For the Subprogram rule: *if its left premise is derivable then its conclusion is derivable* **if and only if** *its right premise is derivable.*

For the Branching rule: *its conclusion is derivable* **if and only if** *both its premises are derivable.*

6.7 Kripke models vs. Computational databases

A database that refutes a propositional formula f can be easily transformed into a Kripke model K that refutes this formula f.

Corollary 6.4 *For every task sequent S, S is derivable in* **the Calculus of Tasks if and only if** *S is derivable in* **Intuitionistic Propositional Logic .**

But there is **no straightforward inverse transformation** because such an inverse **should ensure** the full accordance with all the functional entities, which is a very strong and tough condition.

In particular, considering **self-** and **cross-referential** functional entities and variant dependencies, we **cannot do without the Recursion Theorem and infinite domains.**

The following example is related with the problem of unbounded domains.

Example 6.1

Let us study the well-known sequent

$$\Gamma_1 \vdash C$$

where Γ_1 consists of seven formulas:

Formulas	Comments
$(F \to (B_1 \vee B_2))$,	
$((F \to B_1) \to G_1)$, $(G_1 \& F \to B_1)$,	These two formulas represent the functional G_1 of the type $(F \to B_1)$
$((F \to B_2) \to G_2)$, $(G_2 \& F \to B_2)$,	These two formulas represent the functional G_2 of the type $(F \to B_2)$
$(G_1 \to C)$, $(G_2 \to C)$.	

It should be noted that this sequent is not derivable in **Intuitionistic Propositional Logic** .

Similarly to Example 4.2,

$$\Gamma_1 = Axioms(KB_1)$$

for the following knowledge base

$$KB_1 = (Names_1, Functs_1, Doms_1, Deps_1)$$

where

- $Names_1 = (B_1, B_2, C, F, G_1, G_2)$
- $Functs_1$ consists of two functional entities:
  - (1) G_1 of the type $(F \to B_1)$, and
  - (2) G_2 of the type $(F \to B_2)$.
- $Deps_1$ consists of three formulas:

$$(F \to (B_1 \vee B_2)), \quad (G_1 \to C), \quad (G_2 \to C).$$

Lemma 6.17 *Let $Dom(F)$ be a singleton.*

If the variant dependency $(F \to (B_1 \vee B_2))$ is satisfied on a database T then either functional dependency $(F \to B_1)$ or functional dependency $(F \to B_2)$ is satisfied on the database T.

Corollary 6.5 *If $Dom(F)$ is a singleton then this sequent*

$$Axioms(KB_1) \vdash C$$

is to be solvable.

This result can be generalized to demonstrate that there is no uniform upper bound for domains:

Corollary 6.6 *For each integer n, we can construct a knowledge base KB_n such that*

- *For every A, the set $Dom(A)$ contains at least n elements.*
- *There is a task sequent of the form*

$$Axioms(KB_n) \vdash Z$$

such that

(i) *it is solvable in each of the senses of Theorem 5.1, but*

(ii) *it is not derivable in* **Intuitionistic Propositional Logic** .

The following example shows **the combinatorial troubles that have been circumvented** by using a version of the Recursion Theorem represented in Lemma 6.8.

Example 6.2 *(continuing Example 6.1)*
Now, we consider the sequent

$$\Gamma^1 \vdash C$$

where Γ^1 is the following set of ten formulas:

Formulas	Comments
Γ_1,	from Example 6.1
$((A \to F) \to F)$, $(F\&A \to F)$,	Represent the **self-reproducible** functional F of the type $(A \to F)$
$(\emptyset \to A)$.	

It should be also noted that this new sequent is not derivable in **Intuitionistic Propositional Logic** .

We have that

$$\Gamma^1 = Axioms(KB^1)$$

for the knowledge base

$$KB^1 = (Names^1, Functs^1, Doms^1, Deps^1)$$

where

- $Names^1 = (A, B_1, B_2, C, F, G_1, G_2)$
- $Functs^1$ consists of three functional entities:

  (1) G_1 of the type $(F \to B_1)$,
  (2) G_2 of the type $(F \to B_2)$, and
  (3) F of the type $(A \to F)$.

- $Deps^1$ consists of four formulas:

$$(F \to (B_1 \vee B_2)),\ (G_1 \to C),\ (G_2 \to C),\ (\emptyset \to A).$$

Lemma 6.18 *Let a database T be consistent with KB^1.*

If $(\emptyset \to C)$ is not satisfied on T then there are two states s_1 and s_2 in T such that, for some integers a, p_1 and p_2,

(I) $F(s_1) = p_1$,

(II) $F(s_2) = p_2$,

(III) $A(s_1) = A(s_2) = a$,

(IV) φ_{p_1} *and* φ_{p_2} *are different functions,*

(V) *for these* **different fixed points** p_1 *and* p_2, *the following system of equations holds:*

$$\begin{cases} \varphi_{p_1}(a) &= p_1 \\ \varphi_{p_2}(a) &= p_2 \end{cases}$$

Proof. According Lemma 6.17, there are two states s_1 and s_2 such that, for some integer a and different integers p_1 and p_2,

(I) $F(s_1)$ is defined and $F(s_1) = p_1$,

(II) $F(s_2)$ is defined and $F(s_2) = p_2$,

(III) $A(s_1) = A(s_2) = a$.

By item (d) of Definition 5.4, we have:

$$\begin{aligned} \varphi_{p_1}(a) &= p_1, \\ \varphi_{p_2}(a) &= p_2. \end{aligned}$$

■

7 Complexity of Intuitionistic Propositional Logic

7.1 The Algorithm of Analysis and Synthesis

Let us consider the following algorithm based on **the Calculus of Tasks** .

Input. A task sequent $S : \Gamma \vdash Z$.

Output. A scheme program for S if S is solvable, or a refutable database, otherwise.

Method. A derivation of the sequent $\Gamma \vdash Z$ is being searched for.

If this search is successful then, by Corollary 6.1, the derivation is transformed into the scheme program for S.

Otherwise, the answer is: "S is unsolvable" and the corresponding refutable database is constructed with the help of Theorem 6.2.

Corollary 7.1 *Our algorithm runs correctly on all finite task sequents.*

7.2 Space Complexity of Intuitionistic Logic

Theorem 7.1 *For a given sequent S, one can*

(a) recognize whether S is solvable or not,

(b) construct a minimal scheme program for S,

in deterministic **linear** *space*

$$O(L)$$

where L is the number of all occurrences of literals in S.

Proof. Our algorithm can search for a derivation of the sequent $\Gamma \vdash Z$ with the help of a depth-first search.

Taking into account Corollary 6.3, for a given sequent $\Gamma_1 \vdash Z_1$ that is assigned to a vertex of this search tree, the search stack needs to contain no more than

(1) a number of names of 'variant' and 'operator' formulas (without repetitions),

(2) the set $Out(\Gamma_1)$ (without repetitions).

Thus we get a linear bound on the stack size. ∎

7.3 Subtasks and Measure of Unnaturalness

Taking into account the PSPACE-completeness, all known provers are forced to perform an exponential search for **"almost all"** task sequents. In spite of this, all examples of "bad" tasks are unnatural.

We introduce some level (rank) r to task sequents, so that natural (realistic) tasks have a small level r.

Now let us explain what subtasks are and how they interact.

According to what has been said, in performing a task

$$\Gamma \vdash Z$$

there may appear **subtasks,**
e.g. in such cases as follows:

(a) For a functional entity F of the type $(U \to V)$,
we must solve a subtask

$$\Gamma,\ (\emptyset \to U) \vdash V\ ,$$

the input of it is the input-of-functional U.

(b) If we use a variant dependency $(\emptyset \to (Y_1 \vee Y_2))$ for computing some Z_1,
we have to solve two subtasks

$$\Gamma,\ (\emptyset \to Y_1) \vdash Z_1$$

and

$$\Gamma,\ (\emptyset \to Y_2) \vdash Z_1\ ,$$

where the inputs of these subtasks are the alternatives Y_1 and Y_2.

In performing the main task, subtasks can interact, namely, we can solve a subtask provided that values of inputs of some other subtasks are given in addition. In particular, embedding of subprograms is related to this phenomenon (Cf. Theorem 7.3).

Definition 7.1 (The degree of subtask interaction)
For a given task sequent S, we say that **the degree of subtask interaction in** S is not greater than r if there is a solution for S such that the maximal number of subtasks with different inputs which can interact in the process of this solution does not exceed the integer r. [16]

7.4 Non-uniform upper bound for Time Complexity

Theorem 7.2 (**[Kanovich87, Kanovich91]**) *For a given sequent*

$$S: \Gamma \vdash Z \ ,$$

one can

(a) recognize whether S is solvable or not, and

(b) construct a program for S,

in **quasipolynomial** *running time*

$$O(\frac{L \cdot n^2 \cdot m^{3r}}{(r!)^3})$$

where	L	*is the number of all occurrences of literals in* Γ,
	n	*is the number of different literals from* Γ,
	m	*is the total number of* **different** *input-of-functionals and alternatives from* Γ,
	r	*is the minimal degree of subtasks interaction with which S can be solved.*

Proof. Due to the limitations of space, I give only the gist of the proof, the detailed proof will appear in a subsequent paper.

We found it convenient to use the concept of labelled deductive systems of Dov Gabbay [Gabbay89]. The main idea is to modify **the Calculus of Tasks** and to labelize it so that we can control the process of interaction of those subtasks the inputs of which belong to a certain fixed set. ∎

In fact, our algorithm runs faster than in Theorem 7.2.

Let us consider small r.

[16] Formal definitions are in [Kanovich87, Kanovich91].

Corollary 7.2 *For all sequents with* $r = 0$ *(that corresponds to Horn clauses), our algorithm runs in* **linear time.**

If the minimal degree of subtasks interaction with which a task can be solved is equal to 1, we say that this task is solvable **with separable subtasks.**

Corollary 7.3 ([DK85]) *Our algorithm runs in* **quadratic time** *for all task sequents that can be solved with separable subtasks.*

Corollary 7.4 *We can solve task sequents with separable subtasks in parallel near-linear time.*

7.5 Subtask Interaction vs. Embedding of Subtasks

Finally, let $Task_{Embedding=r}$ be a set of all tasks for which there exist programs such that embedding of their subprograms and conditional statements is not greater than r.

Theorem 7.3 *For every* r*, this class is a* **proper** *subclass of the class of all tasks that are solvable with degree* r *of subtasks interaction.*

On the other hand, Theorem 5.1 implies

Corollary 7.5 *For each entity* A*, let* $Dom(A)$ *be either infinite or empty.*

Then a task $\Gamma \vdash Z$ *is solvable* **if and only if** *it is contained in the class* $Task_{Embedding=m}$*,*
where m *is the total number of different input-of-functionals and alternatives from* Γ*.*

Theorem 7.4 *For a given task sequent* $S:\ \Gamma \vdash Z$ *, we can*

(a) recognize whether S *is solvable or not,*

(b) construct a program for S*,*

in **quasipolynomial** *running time*

$$O(L \cdot n \cdot m^r)$$

in **linear** *space*

$$O(L)$$

where r *is the minimal integer such that* S *is contained in the class* $Task_{Embedding=r}$*.*

Theorem 7.5 *For every r, the class $Task_{Embedding=r}$ is running in parallel near-linear time.*

Summary: Treating task sequents on the basis of our calculus, an exponential execution time should be expected in the worst case, as is customary.

But such cases arise for very unnatural tasks that need maximum cross-linking of all possible subprograms, even in the best programs.

Our prover runs in polynomial time on all natural tasks; the degree of the polynomial is determined by the minimal depth of interacting of subtasks that can be achieved in the best solutions for the main task.

8 Finite vs. Infinite Knowledge Bases

It should be pointed out that Theorem 5.1 is valid for all infinite knowledge bases containing no variant dependencies, as well as for many infinite knowledge bases having such dependencies.

Nevertheless, we can show an infinite knowledge base KB (containing only a single variant dependency) and a task sequent

$$S : \ Axioms(KB) \vdash Z$$

such that

(1) this S is solvable in each of the senses of Theorem 5.1, but still

(2) there is no program for this S.

9 Conclusion

In conclusion it should be pointed out that

- In fact, our calculus works as **a rewriting system,** by means of **simplifying tasks and reducing them to equivalent** *normal* **forms.**
- On the basis of similar **information preserving** calculi we can get algorithms that **run in polynomial (and even linear or subquadratic) time** also for

(1) the membership problem in the theory of relational databases with functional and multivalued dependencies,

(2) recognizing the validity of Horn formulas in monadic predicate logic,

(3) flow analysis of "and-or" graphs,

(4) recognizing derivability of formulas of some kind in the classical and intuitionistic propositional and modal calculi, etc.

Acknowledgements

I would like to thank the participants of Sergei Artemov's seminar on Logical Methods in Computer Science at the Moscow State University for the opportunity to develop and clarify our results.

I am greatly indebted to my Dutch colleagues: Johan van Benthem, Jan van Eijck, Dick de Jongh, Michiel van Lambalgen, Fer-Jan de Vries, Anne Troelstra and many others from the University of Amsterdam and CWI for very useful and stimulating discussions.

References

[AHU76] A.Aho, J.Hopcroft and J.Ullman, The Design and Analysis of Computer Algorithms, (1976).

[DK85] A.Ja.Dikovskii and M.I.Kanovich, Computational models with separable subtasks. Proceedings of Academy of Sci. of USSR, Technical Cybernetics, 5 (1985), 36-60. (Russian)

[Gabbay89] D.M.Gabbay, Labelled deductive systems. 1st draft September 1989, To appear as a book.

[GJ79] M.R.Garey and D.S.Johnson, Computers and Intractability, (1979).

[Jongh68] D.H.J. de Jongh. Investigations on the intuitionistic propositional calculus. Ph.D.Thesis, University of Wisconsin, Madison, 1968.

[Kanovich87] M.I.Kanovich, Quasipolynomial algorithms for recognizing the satisfiability and derivability of propositional formulas. Soviet Mathematics Doklady, 34, N 2 (1987), 273-277.

[Kanovich90] M.I.Kanovich, Efficient program synthesis in computational models. J. Logic Programming, 9, N 2-3 (1990), 159-177.

[Kanovich91] M.I.Kanovich, Efficient program synthesis: Semantics, Logic, Complexity. Theoretical Aspects of Computer Software, TACS'91, Japan, Sendai, 1991, September.

[Ladner77] R.Ladner, The computational complexity of provability in systems of modal propositional logic. SIAM J. Computing, 6 (1977), 467-480.

[Mints83] G.E.Mints and E.Kh.Tyugu, Third All-Union Conference 'Application of the Methods of Mathematical Logic', Proceedings, Tallinn, (1983), 52-60. (Russian)

[Mints90] G.E.Mints and E.Kh.Tyugu, Propositional logic programming and the PRIZ system. J. Logic Programming, 9, N 2-3 (1990), 179-193.

[Rogers67] H.Rogers, Theory of Recursive Functions and Effective Computability, (1967).

[Statman79] R.Statman, Intuitionistic propositional logic is Polynomial-Space complete. Theoret. Computer Sci., 9 (1979), 67-72.

[Ullman80] J.D.Ullman, Principles of Database Systems, (1980).

The ILLC Prepublication Series

1990 *Logic, Semantics and Philosophy of Language*
LP-90-01 Jaap van der Does A Generalized Quantifier Logic for Naked Infinitives
LP-90-02 Jeroen Groenendijk, Martin Stokhof Dynamic Montague Grammar
LP-90-03 Renate Bartsch Concept Formation and Concept Composition
LP-90-04 Aarne Ranta Intuitionistic Categorial Grammar
LP-90-05 Patrick Blackburn Nominal Tense Logic
LP-90-06 Gennaro Chierchia The Variablity of Impersonal Subjects
LP-90-07 Gennaro Chierchia Anaphora and Dynamic Logic
LP-90-08 Herman Hendriks Flexible Montague Grammar
LP-90-09 Paul Dekker The Scope of Negation in Discourse, towards a Flexible Dynamic Montague grammar
LP-90-10 Theo M.V. Janssen Models for Discourse Markers
LP-90-11 Johan van Benthem General Dynamics
LP-90-12 Serge Lapierre A Functional Partial Semantics for Intensional Logic
LP-90-13 Zhisheng Huang Logics for Belief Dependence
LP-90-14 Jeroen Groenendijk, Martin Stokhof Two Theories of Dynamic Semantics
LP-90-15 Maarten de Rijke The Modal Logic of Inequality
LP-90-16 Zhisheng Huang, Karen Kwast Awareness, Negation and Logical Omniscience
LP-90-17 Paul Dekker Existential Disclosure, Implicit Arguments in Dynamic Semantics
Mathematical Logic and Foundations
ML-90-01 Harold Schellinx Isomorphisms and Non-Isomorphisms of Graph Models
ML-90-02 Jaap van Oosten A Semantical Proof of De Jongh's Theorem
ML-90-03 Yde Venema Relational Games
ML-90-04 Maarten de Rijke Unary Interpretability Logic
ML-90-05 Domenico Zambella Sequences with Simple Initial Segments
ML-90-06 Jaap van Oosten Extension of Lifschitz' Realizability to Higher Order Arithmetic, and a Solution to a Problem of F. Richman
ML-90-07 Maarten de Rijke A Note on the Interpretability Logic of Finitely Axiomatized Theories
ML-90-08 Harold Schellinx Some Syntactical Observations on Linear Logic
ML-90-09 Dick de Jongh, Duccio Pianigiani Solution of a Problem of David Guaspari
ML-90-10 Michiel van Lambalgen Randomness in Set Theory
ML-90-11 Paul C. Gilmore The Consistency of an Extended NaDSet
Computation and Complexity Theory
CT-90-01 John Tromp, Peter van Emde Boas Associative Storage Modification Machines
CT-90-02 Sieger van Denneheuvel, Gerard R. Renardel de Lavalette A Normal Form for PCSJ Expressions
CT-90-03 Ricard Gavaldà, Leen Torenvliet, Osamu Watanabe, José L. Balcázar Generalized Kolmogorov Complexity in Relativized Separations
CT-90-04 Harry Buhrman, Edith Spaan, Leen Torenvliet Bounded Reductions
CT-90-05 Sieger van Denneheuvel, Karen Kwast Efficient Normalization of Database and Constraint Expressions
CT-90-06 Michiel Smid, Peter van Emde Boas Dynamic Data Structures on Multiple Storage Media, a Tutorial
CT-90-07 Kees Doets Greatest Fixed Points of Logic Programs
CT-90-08 Fred de Geus, Ernest Rotterdam, Sieger van Denneheuvel, Peter van Emde Boas Physiological Modelling using RL
CT-90-09 Roel de Vrijer Unique Normal Forms for Combinatory Logic with Parallel Conditional, a case study in Conditional Rewriting
Other Prepublications
X-90-01 A.S. Troelstra Remarks on Intuitionism and the Philosophy of Mathematics, Revised Version
X-90-02 Maarten de Rijke Some Chapters on Interpretability Logic
X-90-03 L.D. Beklemishev On the Complexity of Arithmetical Interpretations of Modal Formulae
X-90-04 Annual Report 1989
X-90-05 Valentin Shehtman Derived Sets in Euclidean Spaces and Modal Logic
X-90-06 Valentin Goranko, Solomon Passy Using the Universal Modality: Gains and Questions
X-90-07 V.Yu. Shavrukov The Lindenbaum Fixed Point Algebra is Undecidable
X-90-08 L.D. Beklemishev Provability Logics for Natural Turing Progressions of Arithmetical Theories
X-90-09 V.Yu. Shavrukov On Rosser's Provability Predicate
X-90-10 Sieger van Denneheuvel, Peter van Emde Boas An Overview of the Rule Language RL/1
X-90-11 Alessandra Carbone Provable Fixed points in $I\Delta_0+\Omega_1$, revised version
X-90-12 Maarten de Rijke Bi-Unary Interpretability Logic
X-90-13 K.N. Ignatiev Dzhaparidze's Polymodal Logic: Arithmetical Completeness, Fixed Point Property, Craig's Property
X-90-14 L.A. Chagrova Undecidable Problems in Correspondence Theory
X-90-15 A.S. Troelstra Lectures on Linear Logic

1991 *Logic, Semantics and Philosophy of Language*
LP-91-01 Wiebe van der Hoek, Maarten de Rijke Generalized Quantifiers and Modal Logic
LP-91-02 Frank Veltman Defaults in Update Semantics
LP-91-03 Willem Groeneveld Dynamic Semantics and Circular Propositions
LP-91-04 Makoto Kanazawa The Lambek Calculus enriched with Additional Connectives
LP-91-05 Zhisheng Huang, Peter van Emde Boas The Schoenmakers Paradox: Its Solution in a Belief Dependence Framework
LP-91-06 Zhisheng Huang, Peter van Emde Boas Belief Dependence, Revision and Persistence
LP-91-07 Henk Verkuyl, Jaap van der Does The Semantics of Plural Noun Phrases
LP-91-08 Víctor Sánchez Valencia Categorial Grammar and Natural Reasoning
LP-91-09 Arthur Nieuwendijk Semantics and Comparative Logic
LP-91-10 Johan van Benthem Logic and the Flow of Information
Mathematical Logic and Foundations
ML-91-01 Yde Venema Cylindric Modal Logic
ML-91-02 Alessandro Berarducci, Rineke Verbrugge On the Metamathematics of Weak Theories
ML-91-03 Domenico Zambella On the Proofs of Arithmetical Completeness for Interpretability Logic
ML-91-04 Raymond Hoofman, Harold Schellinx Collapsing Graph Models by Preorders
ML-91-05 A.S. Troelstra History of Constructivism in the Twentieth Century
ML-91-06 Inge Bethke Finite Type Structures within Combinatory Algebras
ML-91-07 Yde Venema Modal Derivation Rules
ML-91-08 Inge Bethke Going Stable in Graph Models
ML-91-09 V.Yu. Shavrukov A Note on the Diagonalizable Algebras of PA and ZF
ML-91-10 Maarten de Rijke, Yde Venema Sahlqvist's Theorem for Boolean Algebras with Operators
ML-91-11 Rineke Verbrugge Feasible Interpretability
ML-91-12 Johan van Benthem Modal Frame Classes, revisited
Computation and Complexity Theory
CT-91-01 Ming Li, Paul M.B. Vitányi Kolmogorov Complexity Arguments in Combinatorics
CT-91-02 Ming Li, John Tromp, Paul M.B. Vitányi How to Share Concurrent Wait-Free Variables
CT-91-03 Ming Li, Paul M.B. Vitányi Average Case Complexity under the Universal Distribution Equals Worst Case Complexity
CT-91-04 Sieger van Denneheuvel, Karen Kwast Weak Equivalence
CT-91-05 Sieger van Denneheuvel, Karen Kwast Weak Equivalence for Constraint Sets
CT-91-06 Edith Spaan Census Techniques on Relativized Space Classes
CT-91-07 Karen L. Kwast The Incomplete Database
CT-91-08 Kees Doets Levationis Laus
CT-91-09 Ming Li, Paul M.B. Vitányi Combinatorial Properties of Finite Sequences with high Kolmogorov Complexity
CT-91-10 John Tromp, Paul Vitányi A Randomized Algorithm for Two-Process Wait-Free Test-and-Set
CT-91-11 Lane A. Hemachandra, Edith Spaan Quasi-Injective Reductions
CT-91-12 Krzysztof R. Apt, Dino Pedreschi Reasoning about Termination of Prolog Programs
Computational Linguistics
CL-91-01 J.C. Scholtes Kohonen Feature Maps in Natural Language Processing
CL-91-02 J.C. Scholtes Neural Nets and their Relevance for Information Retrieval
CL-91-03 Hub Prüst, Remko Scha, Martin van den Berg A Formal Discourse Grammar tackling Verb Phrase Anaphora
Other Prepublications
X-91-01 Alexander Chagrov, Michael Zakharyaschev The Disjunction Property of Intermediate Propositional Logics
X-91-02 Alexander Chagrov, Michael Zakharyaschev On the Undecidability of the Disjunction Property of Intermediate Propositional Logics
X-91-03 V. Yu. Shavrukov Subalgebras of Diagonalizable Algebras of Theories containing Arithmetic
X-91-04 K.N. Ignatiev Partial Conservativity and Modal Logics
X-91-05 Johan van Benthem Temporal Logic
X-91-06 Annual Report 1990
X-91-07 A.S. Troelstra Lectures on Linear Logic, Errata and Supplement
X-91-08 Giorgie Dzhaparidze Logic of Tolerance
X-91-09 L.D. Beklemishev On Bimodal Provability Logics for Π_1-axiomatized Extensions of Arithmetical Theories