Data storage interpretation of labeled modal logic*

Sergei Artëmov[†]
Steklov Mathematical Institute,
Vavilova str. 42,
Moscow, 117966 RUSSIA.
e-mail: sergei@artemov.mian.su

Vladimir Krupski[‡]
Department of Mathematics
Moscow State University
Moscow 119899, RUSSIA
email: krupski@sci.math.msu.su

February 10, 1995

Abstract

We introduce *reference structures* – a basic mathematical model of a data organization capable to store and utilize information about its addresses. A propositional labeled modal language is used as a specification and programming language for reference structures; the satisfiability algorithm for modal language gives a method of building and optimizing reference structures satisfying a given formula. Corresponding labeled modal logics are presented, supplied with cut free axiomatizations, completeness and decidability theorems are proved. Initialization of typed variables in some programming languages is presented as an example of a reference structure building.

1 Introduction

We suggest to interpret a labeled modal formula $[\![m]\!]A$ as "memory cell m stores sentence A" and to treat propositional variables as names of the cell contents. The

**Annals of Pure and Applied Logic*, v. 78, pp. 57-71, 1996

[†]The research described in this publication was made possible in part by Grant No.NFQ000 from the International Science Foundation and by Grant No.93-011-16015 of the Russian Foundation for Fundamental Research.

[‡]Partially supported by the grant No.95-01-00416 of the Russian Foundation for Fundamental Research.

labeled modal language allows to keep control over both unification of names and validity of the information stored.

All this eventually makes it possible to do some sort of programming of referential data structures by means of labeled modal language in the following way.

We consider a language with

- atomic data constants $c_1, c_2, \ldots$,
- variables $m_1, m_2, \ldots$ for memory cell addresses,
- operation $\hat{}$ of reading the contents of a cell, operation $[\![\cdot]\!](\cdot)$ for storing information to a cell, boolean connectives.

A formula in this language may be regarded as a specification of a memory configuration which stores data files $c_1, c_2, \ldots$ together with an information about contents of other cells, location of files, etc. The standard completeness and cut elimination proof of a corresponding *logic of refence structures* in fact gives an algorithm which verifies the unifiability of names and semantical correctness of this specification and in a positive case provides a data allocation table in abstract addresses.

The compiling problem turns out to be *NP*-complete. The corresponding algorithm suggested in the current paper is a hybrid of the unification and some sort of boolean satisfiability procedures.

The restriction of the underlying objects to sentences (with *validity relation* on them) does not lead to a loss of generality for our purposes: if a proper data c_i originally represents a number N, we assume that c_i is the sentence "this is a number N"; the same treatment may be given to other sorts of proper information: terms, names, addresses, etc.

The general definition of a reference structure covers not only a wide class of computer data organizations, but also cross-references with built in reference assignments in formal languages, the system of proofs and theorems in a formal theory, etc. However in the current paper we restict this general definition to *pure reference structures* closely oriented to the computer data bases. Since there will be no others here we will use a general name *reference structures* for the pure ones.

2 Reference Structures

2.1 Definition. The language $\mathcal{L}(M, C)$ of a reference structure depends on two sets M and C and is defined as follows. Let $C = \{c_1, c_2, \ldots, \top\}$ be a set of *data constants*, which will represent a proper information to be stored in a reference structure. Let also $M = \{1, 2, \ldots\}$ be a set of *memory cells*. The language of a reference structure contains *storage operators* $[\![1]\!](\cdot), [\![2]\!](\cdot), \ldots$, one for each memory cell, together with usual boolean connectives $\{\wedge, \vee, \neg, \top\}$. For any cell $m \in M$ there is a *reference variable* v_m, $v_m \equiv \widehat{m}$ for short. One should not be mislead by the notation: $\widehat{1}, \widehat{2} \ldots$ are indeed variables, not constants, since the reading function corresponding to $\widehat{\ }$ will itself be a parameter of a reference structure. We denote by V the set of all *reference variables*. The set of *formulas* $Fm(M, C)$ is the least set such that

$C, V \subseteq Fm(M, C)$,

if $m \in M$ and $A \in Fm(M, C)$, then $[\![m]\!]A \in Fm(M, C)$,

if $A, B \in Fm(M, C)$, then $(A \wedge B), (A \vee B), (A \to B), (\neg A) \in Fm(M, C)$.

2.2 Definition. A formula is *ground* if it contains no reference variables, under $St(M, C)$ we mean the set of all ground formulas of the language $\mathcal{L}(M, C)$. A *substitution* is a partial mapping $\theta : V \longrightarrow Fm(M, C)$; θ is a solution of an equation $A = B$, for $A, B \in Fm(M, C)$, if $A\theta = B\theta$. Substitution θ is a solution of a relation $R \subseteq M \times Fm(M, C)$ if θ is a solution of $\widehat{m} = A$ for every $(m, A) \in R$. A substitution $\theta : V \longrightarrow St(M, C)$ is called *ground substitution*. We assume that all atom constants are valid. Any ground solution θ of a relation $R \subseteq M \times Fm(M, C)$ naturally defines a validity relation $\models_{R,\theta}$ on all ground formulas:

$\models_{R,\theta} \top$ and $\models_{R,\theta} c$ for all $c \in C$,

$\models_{R,\theta} [\![m]\!]A \;\Leftrightarrow\; (m \in Dom\ R$ and $\widehat{m}\theta = A)$,

$\models_{R,\theta}$ respects boolean connectives.

A ground solution θ of a relation R is valid on $N \subseteq M$, if $\models_{R,\theta} \widehat{n}\theta$ for all $n \in N$. A *storage table* is a functional relation $R \subseteq M \times Fm(M, C)$ between memory cells and formulas.

2.3 Definition. A *reference structure* is a storage table which has a ground solution θ valid on $Dom\ R$, i.e. a storage table with all stored sentences to be true.

2.4 Comment. The relation R *is* a system of assigning memory cells to formulas from $Fm(M, C)$ which is consistent from both combinatorial and semantical sides.

The cells which are not in *Dom* R are called *empty.* The reserve of empty cells is both realistic and technically convenient. If R has a ground solution satisfying the definition of a reference structure above, then R has such a solution which is total on V. Without loss of generality we assume that θ is already total and call it a *reading procedure* of R. A reading procedure provides a ground picture of the cell contents where all the references are already given their "real" meaning in terms of proper information and storage connections. On empty cells a reading procedure returns some ground sentences which may be regarded as sort of "error messages".

2.5 Definition. Let R be a reference structure and θ – its reading procedure. With a pair $\Re = (R, \theta)$ we associate a validity relation $\models$ defined on all formulas from $Fm(M, C)$:

$$\Re \models A \quad \Leftrightarrow \quad \models_{R,\theta} A\theta.$$

It is easy to see that $\models$ is an extention of the "old" validity relation $\models_{R,\theta}$ from $St(M, C)$ to $Fm(M, C)$. Also, $\Re \models A$ for all $A \in$ *Val* R.

2.6 Lemma.

$$\Re \models [\![m]\!]A \quad \Rightarrow \quad \Re \models A.$$

Proof follows immediately from the definitions.

2.7 Lemma. *The following are equivalent*

1. *m is nonempty,*
2. *$\Re \models [\![m]\!]B$ for some formula B,*
3. *$\Re \models [\![m]\!]\widehat{m}$.*

Proof. 3. $\Rightarrow$ 2. $\Rightarrow$ 1. are trivial. We prove the remaining 1. $\Rightarrow$ 3. If $m \in$ *Dom* R, then $(m, A) \in R$ for some formula A $\widehat{m}\theta = A\theta$, hence $\models_{R,\theta} [\![m]\!]A\theta$ and $\Re \models [\![m]\!]A$. ■

We can see now how the decision to have a reserve of empty cells increases the expressive power of the reference structures language. For example, the fact that $m \in M$ is not empty can now be expressed by a formula $[\![m]\!]\widehat{m}$, which we will denote $\widetilde{m}$ and will use as a natural sentence format pointer. The meaning of $\widetilde{m}$ as a pointer is assumed to be built in the search algorithm. Note that the length of $\widetilde{m}$ can be easily made of the order of the length of m and $\widehat{m}$. i.e. "very small".

2.8 Example. A *list* of $c_1, c_2, \ldots, c_n$ may be described as the ground reference structure R over $M = \{1, 2, \ldots, n+1\}$ and $C = \{c_1, c_2, ..., c_n\}$ as

$$R = \{(1, \varphi_1), \ldots, (n, \varphi_n), (n+1, \top)\}$$

for $\varphi_1 = A_1 \wedge [\![2]\!]\varphi_2$, $\varphi_2 = A_2 \wedge [\![3]\!]\varphi_3, \ldots,$ $\varphi_n = A_n \wedge [\![n+1]\!]\top$. Here $\top$ works as a marker of the end node. The list can be represented by the formula $[\![1]\!]\varphi_1$).

It does not mean, however that we intend to store the entire list in one cell 1. We will see now how a regular reference structure "list" looks like:

$$\widehat{R} = \{(i, A_1 \wedge \widetilde{2}), (2, A_2 \wedge \widetilde{3}), \ldots, (n, A_n \wedge \widetilde{n+1}), (n+1, \top)\}.$$

The entire reference structure can now be represented by the formula

$$[\![1]\!](A_1 \wedge \widetilde{2})) \bigwedge \ldots \bigwedge [\![n]\!](A_n \wedge \widetilde{n+1}) \bigwedge [\![n+1]\!]\top.$$

The main question here is how to decide whether there exists a reference structure satisfying given storage description, and to construct one if it exists. A finite equation system alone can be solved in linear time (cf.[4], [5]). The semantic component however spoils the picture: the problem immediately becomes at least *NP*-hard, since it naturally includes the satisfiability problem for the classical propositional logic. Below we'll show that it is *NP*-complete.

3 Logic of reference structures

Usually the Unification Algorithm deals with finite systems of "unconditional" equalities of the form $A = B$. Fast algorithms of solving such systems were suggested in [4] (cf. also [6]). We assume that formulas are presented as *directed acyclic graphes with shared variables (dags)* which allow lineartime unification ([4]).

We will also be interested in the "conditional" equalities of the form

$$\begin{cases} U_i = V_i, & i \in I \\ S_j = W_j \Rightarrow U_j = V_j, & j \in J \end{cases} \tag{1}$$

For a convenience we consider some deterministic variant of the Unification Algorithm by fixing an order of the equations for this algorithm to choose. The suitable modification **U** of the unification algorithm for "conditional" equalities works as follows. Using the standard unification algorithm solve the unconditional part of the system and calculate its m.g.u. σ. Then pick a "conditional" equality and check the conditions

$$S_j\sigma = W_j\sigma.$$

If the condition fails, then take the next "conditional" equality. If the conditions are fulfilled, add the succedent equality to the unconditional part and solve the system again. The process terminates when the checking procedure fails to add new equalities or the unification algorithm fails to solve a current unconditional part of the system. The standard argument proves that this modification gives the most general unifier (m.g.u.) of the system with "conditional" equations. The *standard* m.g.u. of the set of equations (1) is the m.g.u. obtained by $\mathbf{U}$.

3.1 Lemma. (cf. [2]). *Let σ be the standard m.g.u. of a "conditional" system (1). Then*

1. *all variables occurring in σ are from (1),*
2. *$Dom(\sigma) \cap Val(\sigma) = \emptyset$,*
3. *σ is idempotent, i.e. $\sigma \circ \sigma = \sigma$,*
4. *for every solution θ of (1) there exists a substitution τ s.t. $\theta = \sigma \circ \tau$.*

Consider a labeled modal language $\widehat{L}$ which contains

memory cell variables $CVar = \{m_1, m_2, m_3, \ldots\}$,

reference variables $RVar = \{\widehat{m_1}, \widehat{m_2}, \widehat{m_3} \ldots\}$,

sentence constants $Con = \{c_1, c_2, c_3, \ldots\}$, the truth constant $\top$

and is closed under boolean connectives and *labeled modalities* $[\![m_i]\!](\cdot), i = 1, 2, \ldots$ (unary operators).

The difference between $\widehat{L}$ and $\mathcal{L}(M, C)$ is that the cell addresses in $\widehat{L}$ are variables, unlike $\mathcal{L}(M, C)$, where they are constants.

3.2 Definition. Let M be a memory set and C a data constants set. An *interpretation* of $\widehat{L}$ to $Fm(M, C)$ is a mapping $*$ of $CVar$ into M and Con into C which is injective on Con. The interpretation $*$ has a canonical extension to all $\widehat{L}$ fromulas:

$\top^* = \top$,

for $\widehat{p} \in RVar \quad \widehat{p}^{\,*} = \widehat{p^*}$,

$*$ commutes with the boolean connectives,

$([\![p]\!]A)^*$ is $[\![p^*]\!]A^*$.

We say that a $\widehat{L}$ formula F is *valid* in a reference structure $\Re = (R, \theta)$ under interpretation $*$, if $\Re \models F^*$. A reference structure $\Re$ is a *model* of a given set Γ of $\widehat{L}$ formulas under given interpretation $*$ if $\Re \models A^*$ for each $A \in \Gamma$.

The language $\widehat{L}$ may now be regarded as a programming language for reference structures. Here a program is a modal formula A describing the properties of a reference structure $\Re$. Satisfiability of A means the existence of a desired reference structure. The satisfiability algorithm for the language $\widehat{L}$ naturally arises from the completeness proof of the calculus $\mathcal{LR}$ (below).

A substitution on the $\widehat{L}$ formulas works simultaneously in two formats: cells and sentences. No special restrictions on substitutions are imposed. For example, an reference variable can be substituted by any $\widehat{L}$ formula.

Without a loss of generality we restrict the set of cell variables $CVar$ to its finite fragment $\{m_1, m_2, \ldots, m_r\}$ (corresponding restriction should be put on the set of reference variables). Also we assume that *Con* is finite.

3.3 Definition. Under $\sigma_{A,B,p}$ we mean the *standard* m.g.u. of the set of equations

$$\begin{gathered} \widehat{p} = A = B \\ m_i = m_j \Rightarrow \widehat{m_i} = \widehat{m}_j. \end{gathered} \tag{2}$$

Here the "conditional" part is standard with m_i, m_j range over all cell variables occurring in "unconditional" part $\widehat{p} = A = B$. Note that $\sigma_{A,B,p}$ is an idempotent and acts on the variables of all sorts, $m_i\sigma_{A,B,p}$ is a cell variable and $\widehat{m_i}\sigma_{A,B,p}$ is a formula from $\widehat{L}$.

3.4 Definition. We define $C = D \ (\mathrm{mod}\ \ \widehat{p} = A = B)$ to stand for

$$\text{“}C\sigma \equiv D\sigma \ \text{ for every solution } \sigma \text{ of (2)”}.$$

Apparently, if the system (2) has no solution, then $C = D \ (\mathrm{mod}\ \widehat{p} = A = B)$ holds for all C and D. If the system (2) has a solution then

$$C = D \ (\mathrm{mod}\ \ \widehat{p} = A = B) \quad \Leftrightarrow \quad C\sigma_{A,B,p} \equiv D\sigma_{A,B,p}.$$

So, the relation $C = D \ (\mathrm{mod}\ \widehat{p} = A = B)$ is decidable.

Axioms of $\mathcal{LR}$:

(A1) The classical propositional axioms together with constants $\{c_1, c_2, c_3, \ldots, \top\}$ adopted as new axioms,

(A2) $[\![p]\!]A \to A$,

(A3) $[\![p]\!]A \wedge [\![p]\!]B \to (C \to D)$ if $C = D \,(\mathrm{mod}\ \widehat{p} = A = B)$.

Rule *modus ponens.*

Axiom (A3) is similar to the unification axiom from [1] and the functionality axiom from [3].

3.5 Example. The following is provable in $\mathcal{LR}$:

- $\neg([\![p_1]\!]A_1 \wedge \ldots \wedge [\![p_n]\!]A_n)$ if the system

$$\begin{cases} \widehat{p}_k = A_k \quad (k = 1, \ldots, n); \\ p = q \Rightarrow \widehat{p} = \widehat{q} \ \text{ for all cell variables } p, q \\ \qquad\qquad\qquad \text{occurring in } [\![p_1]\!]A_1 \wedge \ldots \wedge [\![p_n]\!]A_n \end{cases} \tag{3}$$

  is not unifiable.

- $[\![p_1]\!]A_1 \wedge \ldots \wedge [\![p_n]\!]A_n \to (B \leftrightarrow C)$ if $B\sigma = C\sigma$ for the most general unifier σ satisfying the condition (3).

4 Completeness theorem

4.1 Lemma. *For any modal formula F if $\mathcal{LR} \vdash F$, then F^* is valid under every interpretation $*$ in reference structures.*

Proof. A straightforward induction on the proof of F. ∎

4.2 Theorem. *For any formula $F \in \widehat{L}$ if $\mathcal{LR} \nvdash F$, then there exists a finite reference structure $\Re$ and interpretation $*$ of the language $\widehat{L}$ into $\Re$ such that $\Re \not\models F^*$.*

Now we introduce a Gentzen style formulation of $\mathcal{LR}$ and prove simultaneously the completeness theorem along with the cut elimination property of the relevant Gentzen style system.

In what follows a *sequent* is a formal expression of the form $\Gamma \supset \Delta$, where Γ and Δ are finite sets of $\widehat{L}$ formulas.

4.3 Definition. $\mathcal{LR}_G$ is the following sequent calculus:

Axioms:

- $\Gamma \supset \Delta$ such that $\Gamma \cap \Delta \neq \emptyset$ or $\top \in \Delta$ or $c \in \Delta$ for some $c \in Con$.
- $\Gamma \supset \Delta$ such that $\Xi \subseteq \Gamma$, where $\Xi = \{[\![p_i]\!]A_i \mid i = 1, 2, \ldots\}$ and the system (3) for Ξ is not unifiable.

Rules:

- Classical rules for $\wedge$,$\neg$ and structural rules together with the cut-rule.
- $\dfrac{A, \Gamma \supset \Delta}{[\![p]\!]A, \Gamma \supset \Delta}$
- $\dfrac{\Xi, B\sigma, \Gamma \supset \Delta}{\Xi, B, \Gamma \supset \Delta}$, $\dfrac{\Xi, \Gamma \supset B\sigma, \Delta}{\Xi, \Gamma \supset B, \Delta}$, where $\Xi = \{[\![pi]\!]A_i \mid i = 1, 2, \ldots\}$ and σ is the most general unifier of (3) for Ξ and obtained as a result of the standard unification algorithm $\mathbf{U}$.

4.4 Definition. $\mathcal{LR}_G^-$ is the system $\mathcal{LR}_G$ without the cut rule.

The following lemma claims the soundness of $\mathcal{LR}_G$ w.r.t. $\mathcal{LR}$.

4.5 Lemma. *If* $\mathcal{LR}_G \vdash \Gamma \supset \Delta$, *then* $\mathcal{LR} \vdash \bigwedge \Gamma \to \bigvee \Delta$. [1]

Proof. Standard induction on the complexity of the proof of $\Gamma \supset \Delta$ in $\mathcal{LR}_G$. ∎

4.6 Definition. *Saturation process* is the nondeterministic procedure constructing a *saturation tree* labeled by pairs (sequent, substitution) as follows:

[1] For $\Omega = \{A_1, A_2, \ldots\}$ $\bigwedge \Omega = A_1 \wedge A_2 \wedge \ldots$, and $\bigvee \Omega = A_1 \vee A_2 \vee \ldots$.

Given the sequent $\Gamma_0 \supset \Delta_0$ put

$$\Gamma_0' = \Gamma_0 \cup \{\top\} \cup \{ \text{ the set of all constants, occurring in } \Gamma_0 \supset \Delta_0\},$$

and label the root by $(\Gamma_0' \supset \Delta_0, \epsilon)$, where ϵ is an empty substitution, and try repeatedly to apply the saturation rules while they add to the tree some node with the label sequent different from the label of its parent. The rules can be applied to an arbitrary leaf of the current part of the tree if its label sequent $\Gamma \supset \Delta$ is not an axiom of $\mathcal{LR}_G$; in the formulations of the rules we suppose that such a leaf (a current node) is already chosen and labeled by $(\Gamma \supset \Delta, \sigma)$.

Saturation rules:

Rule 1. If $A \wedge B \in \Gamma$, then add to the tree a son of the current node labeled by $(\Gamma \cup \{A, B\} \supset \Delta, \sigma)$.

Rule 2. If $A \wedge B \in \Delta$, then add to the tree two sons of the current node labeled by $(\Gamma \supset \Delta \cup \{A\}, \sigma)$ and $(\Gamma \supset \Delta \cup \{B\}, \sigma)$.

Rule 3. If $\neg A \in \Gamma$ ($\neg A \in \Delta$), then add to the tree a son of the current node labeled by $(\Gamma \supset \Delta \cup \{A\}, \sigma)$ (correspondingly, $(\Gamma \cup \{A\} \supset \Delta, \sigma)$).

Rule 4. If $\Box_p A \in \Gamma$, then add to the tree a son of the current node labeled by $(\Gamma \cup \{A\} \supset \Delta, \sigma)$.

Rule 5. Call the unification algorithm **U** to get the most general solution σ' of the system (3) where $\{[\![p_i]\!]A_i, i = 1, ..., n\}$ is the list of all formulas of the form $[\![p_i]\!]A$ from Γ. Add to the tree a son of the current node labeled by $(\Gamma\sigma' \supset \Delta\sigma', \sigma\sigma')$.

4.7 Lemma. *If* $(\Gamma \supset \Delta, \sigma)$ *is a label in a saturation tree, then for any variable* v *occurring in* $\Gamma \supset \Delta$ *we have* $v\sigma = v$.

Proof. First of all we notice that none of the variables from $Dom(\sigma)$ occurs in $Val(\sigma)$ since σ is a product of m.g.u.'s each enjoying the properties of lemma 3.1. Consider a step 5. Any variable v occurring in $\Gamma \supset \Delta$ is neither from $Dom(\sigma)$ nor from $Dom(\sigma')$. Thus v is a fixed point of both σ and σ'. ∎

4.8 Corollary. *For any label* $(\Gamma \supset \Delta, \sigma)$ *of the saturation tree any subformula* A *of* $\Gamma \supset \Delta$ *we have* $A\sigma = A$*, (hence* $\Gamma\sigma = \Gamma$ *and* $\Delta\sigma = \Delta$*).*

4.9 Lemma. $\sigma^2 = \sigma$.

Proof. $\mathit{Dom}(\sigma) \cap \mathit{Val}(\sigma) = \emptyset$. ■

4.10 Lemma. *The saturation process terminates.*

Proof. Rules 1-4 do not change the subformulas of the sequent so they can not be applied infinitely many times. Any application of the rule 5 reduces the set of variables occurring in $\Gamma \supset \Delta$, thus any path in a saturation tree is finite and the tree itself is finite. ■

Therefore the saturation process always terminates and computes some saturation tree of a given sequent. We say that the saturation process *succeeds* if it produces a saturation tree with all leafs labeled with axioms; otherwise it *fails*.

4.11 Lemma. *If the saturation process on a given sequent succeeds, then the sequent is provable in* $\mathcal{LR}_G^-$.

Proof. A saturation tree with all leafs labeled by axioms is in fact the tree-like derivation in $\mathcal{LR}_G^-$ of the sequent labeling the root. ■

Suppose the saturation process fails on a sequent $\Gamma_0 \supset \Delta_0$. Then it produces a leaf of the saturation tree labeled by $(\Gamma \supset \Delta, \sigma)$ such that

- $\Gamma_0\sigma \subseteq \Gamma, \Delta_0\sigma \subseteq \Delta, \Gamma \cap \Delta = \emptyset,\ \top \in \Gamma,\ \mathit{Con} \subset \Gamma$;
- if $(A \wedge B) \in \Gamma$, then $A \in \Gamma$ and $B \in \Gamma$;
- if $(A \wedge B) \in \Delta$, then $A \in \Delta$ or $B \in \Delta$;
- if $\neg A \in \Gamma$, then $A \in \Delta$; if$\neg A \in \Delta$, then $A \in \Gamma$;
- if $[\![p]\!]A \in \Gamma$, then $A \in \Gamma$;
- Γ is functional: $[\![p]\!]A \in \Gamma$ and $[\![p]\!]B \in \Gamma$ imply $A = B$.

Now we are ready to define a reference structure $\Re$ and an interpretation $*$ which will eventually become a countermodel for $\Gamma_0 \supset \Delta_0$.

Let M be the set of cell variables occurring in $\Gamma_0 \supset \Delta_0$, which are fixed points of σ, i.e.

$$M = \{m \in CVar \mid m\sigma = m\}.$$

In particular, all cell variables occurring in $\Gamma \supset \Delta$ are in M. Let the set of data constants C be $Con \cup D$, where D is a set of "new constants" corresponding to fixed point reference variables:

$$D = \{d(\widehat{m}) \mid \widehat{m}\sigma = \widehat{m}\}.$$

In particular, every reference variable $\widehat{m}$ occurring in $\Gamma \supset \Delta$ received a corresponding constant $d(\widehat{m})$.

It is clear, that the fixed point reference variables remain sort of parameters of the future reference structure and they can be evaluated in either way. However the rules of the game require them to become ground sentences. The easiest way to ensure it is to introduce special new constants to evaluate these variables.

Now the set of $Fm(M, C)$ is defined.

Put for any fixed point reference variable $\widehat{m}$

$$\lambda(\widehat{m}) = \begin{cases} d(\widehat{m}), & \text{if } \widehat{m} \in \Gamma, \\ \neg d(\widehat{m}), & \text{otherwise,} \end{cases}$$

Now for any subformula A of $\Gamma \supset \Delta$ we define a ground $A^\lambda \in Fm(M, C)$: $c_i^\lambda = c_i$, $\top^\lambda = \top$, $\widehat{m}^\lambda = \lambda(\widehat{m})$, $([\![m]\!]B)^\lambda = [\![m]\!]B^\lambda$. Note that the translation $^\lambda$ is injective since no constants $d(\widehat{m})$ are unified for different $m \in M$. So, we will write $B\lambda$ instead of B^λ understanding λ as the substitution $\{\widehat{m_1}/\lambda(\widehat{m_1}), \widehat{m_2}/\lambda(\widehat{m_2}) \ldots\}$. Put

$$\theta \;=\; \sigma\lambda,$$

and let

$$R = \{(m, A) \mid [\![m]\!]A \in \Gamma\}.$$

We have to establish now that $\Re = (R, \theta)$ is a reference structure. First, θ is clearly a ground substitution. Then it is easy to see that θ unifies R. Indeed, let $(m, A) \in R$, then $[\![m]\!]A \in \Gamma$, and, by the saturation construction $\widehat{m}\sigma = A\sigma$. Thus

$$\widehat{m}\theta = \widehat{m}\sigma\lambda = A\sigma\lambda = A\theta.$$

4.12 Lemma.
$$A \in \Gamma \;\;\Rightarrow\;\; \models_{R,\theta} A\theta,$$
$$A \in \Delta \;\;\Rightarrow\;\; \not\models_{R,\theta} A\theta.$$

Proof. Induction on A. The cases A is a constant $c_i, \top$ as well as the case A is an reference variable $\widehat{m}$ are covered by the definitions of $\Gamma \supset \Delta$ and θ. The cases of boolean connectives are trivial by the construction of the saturated sequent $\Gamma \supset \Delta$. Now
$$[\![m]\!]B \in \Gamma \;\Rightarrow\; (m, B) \in R \;\Rightarrow\; \widehat{m}\theta = B\theta \;\Rightarrow \models_{R,\theta} [\![m]\!]B\theta.$$
Let now $[\![m]\!]B \in \Delta$. If $m \notin Dom(R)$, then clearly $\not\models_{R,\theta} [\![m]\!]B\theta$. Let now $m \in Dom(R)$, i.e. $[\![m]\!]B' \in \Gamma$ for some B', then $\widehat{m}\theta = B'\theta$. If $B\theta = B'\theta$, then $B\sigma = B'\sigma$, since λ is injective, and thus $B = B\sigma = B'\sigma = B'$, which is impossible because $\Gamma \cap \Delta = \emptyset$. ■

4.13 Lemma. $\mathfrak{R} = (R, \theta)$ *is a reference structure.*

Proof. It only remains to check that $\widehat{m}\theta$ is valid on $Dom(R)$. Let $m \in Dom(R)$, then $(m, A\theta) \in R\theta$ and there exists some B such that $[\![m]\!]B \in \Gamma$ and $B\theta = A\theta$. Then, by the saturation property, $B \in \Gamma$, thus, by lemma 4.12 $\models_{R,\theta} B\theta$, i.e. $\models_{R,\theta} A\theta$. ■

Now we define an interpretation $*$ by $m_i^* = m_i\sigma$, and thus $A^* = A\sigma$ for any $\widehat{L}$ formula A. Note that $A\sigma$ is simultaneously a formula of the reference structure $\mathfrak{R}$.

It is almost trivial now that $\mathfrak{R} \not\models (\bigwedge \Gamma_0 \to \bigvee \Delta_0)^*$. Indeed, if $A \in \Gamma_0$, then $A\sigma \in \Gamma$, and $\mathfrak{R} \models A\sigma\theta$, by lemma 4.12, and $\mathfrak{R} \models A^*$. Similarly, if $A \in \Delta_0$, then $\mathfrak{R} \not\models A^*$.

Thus we have established the following: for any sequent $\Gamma \supset \Delta$ in the language $\widehat{L}$
$$\mathcal{LR}_G^- \not\vdash \Gamma \supset \Delta \;\Rightarrow\; \mathfrak{R} \not\models (\bigwedge \Gamma \to \bigvee \Delta)^* \;\Rightarrow\; \mathcal{LR} \not\vdash \bigwedge \Gamma \to \bigvee \Delta \;\Rightarrow\; \mathcal{LR}_G \not\vdash \Gamma \supset \Delta,$$
which together with the trivial
$$\mathcal{LR}_G \not\vdash \Gamma \supset \Delta \;\Rightarrow\; \mathcal{LR}_G^- \not\vdash \Gamma \supset \Delta$$
gives

4.14 Corollary. (Cut elimination for $\mathcal{LR}_G$) $\mathcal{LR}_G^- = \mathcal{LR}_G$

4.15 Corollary. *$\mathcal{LR}_G$ is an adequate Gentzen style formulation of $\mathcal{LR}$.*

4.16 Corollary. *$\mathcal{LR}$ is decidable.*

Let us complete the proof of Theorem 4.2. Fix a formula F satisfying the conditions of the Theorem 4.2. Put $\Gamma_0 = \emptyset$ and $\Delta_0 = \{F\}$. Since $\mathcal{LR}_G^- \not\vdash \Gamma_0 \supset \Delta_0$ the saturation process on the sequent $\Gamma_0 \supset \Delta_0$ fails, and there is a reference structure $\Re$ such that $\Re \not\models \bigwedge \Gamma_0 \to \bigvee \Delta_0$, i.e. $\Re \not\models F$. ∎

A lazy inspection of the completeness proof above demonstrates that the size of a countermodel (in a dags form) of a given $\widehat{L}$ formula A can be made less than cl^4, where l is the length of A, and c fixed.

Also, on the basis of lineartime unification algorithms from [4], [6] one can easily proof the following time complexity bounds for some natural problems in reference structures.

4.17 Theorem.

1. *The problem "whether $\Re = (R, \theta)$ is a reference structure" is polytime.*
2. *The satisfiability problem for the language $\widehat{L}$ is NP-complete.*

5 Reference structures building and optimization.

The language $\widehat{L}$ can now be considered as a programming language for designing reference structures with reading procedures. A program here is a labeled modal formula P describing the properties of some reference structure $\Re$. The satisfiability algorithm extracted from the proofs of the Theorem 4.2 checks whether P is satisfiable and constructs a finite model of P, which is a desired reference structure.

We reduce the problem of constructing a model of P to the problem of constructing a countermodel for the sequent $P \supset$. The saturation algorithm checks whether this sequent is provable and transforms it into the sequent $\Gamma \supset \Delta$ with saturation properties. If saturation succeeds, then $\mathcal{LR} \vdash \neg P$, and thus there is no reference structure satisfying the condition P. If saturation fails, then we have a quadratic of the size of P reference structure $\Re$ and an interpretation $*$ such that P^* is valid in $\Re$.

Let us consider an example of a problem "initialization of typed variables", which comes from some common programming languages like PASCAL, C, etc.

5.1 Example. (Initialization of typed variables). We consider the following variant of commonly used typing system. Let T be a finite set of primitive types with domains D_τ, $\tau \in T$ (the domains are supposed to be decidable but not necessary disjoint). The set of all types $Type$ is constructed from T by the rules:

	Rule :	Domain :
$(Structure)$	$\dfrac{\tau_i \in Type,\ 1 \leq i \leq n}{\{\tau_1, \ldots, \tau_n\} \in Type}$	$D_1 \times \ldots \times D_n$
$(Union)$	$\dfrac{\tau_i \in Type,\ 1 \leq i \leq n}{\{\tau_1; \ldots; \tau_n\} \in Type}$	$D_1 \cup \ldots \cup D_n$
$(Subset)$	$\dfrac{\tau_i \in Type,\ a_i \in D_{\tau_i},\ 1 \leq i \leq n}{Set\_of(a_1\vert\tau_1, \ldots, a_n\vert\tau_n) \in Type}$	$\mathcal{P}\{a_1, \ldots, a_n\}$

The initialization problem: given a type $\tau \in Type$ and an object $a \in \bigcup_{\tau \in Type} D_\tau$ we have to check whether $a \in D_\tau$ and, if it is, to build a data structure which stores a as an object of type τ together with some address which is the value of corresponding pointer.

The basic elements to construct a reference structure from are constants for objects of primitive types. A reference structure is supposed to represent the type structure in a way that provides a direct access to any subobject of a given object.

With the pair (τ, a) we associate a formula $\Phi(\tau, a) \in \widehat{L}$ and a cell variable p in it:

($Primitive\ type$): $\tau \in T$ and $a \in D_\tau$. Then

$$\Phi(\tau, a) = [\![p]\!] c_{\tau,a},$$

where $c_{\tau,a}$ is a data constant.

($Structure$): $\tau = \{\tau_1, ..., \tau_n\}$ and $a = (a_1, ..., a_n)$. Then

$$\Phi(\tau, a) = (\bigwedge_{i=1}^{n} \Phi_i) \wedge [\![p]\!](\widetilde{p}_1 \wedge \ldots \wedge \widetilde{p}_n),$$

where Φ_i is a variant of $\Phi(\tau_i, a_i)$ obtained from it by renaming the variables of the form $q, \widehat{q}$ (so Φ_i and Φ_j for $i \neq j$ do not have common variables) and p_i is the associated cell variable.

($Union$): $\tau = \{\tau_1; \ldots; \tau_n\}$. Then

$$\Phi(\tau, a) = (\bigvee_{i=1}^{n} \Phi_i') \wedge [\![p]\!]\widetilde{q},$$

where $\Phi_i' = \Phi_i[q/p_i]$, q is a new variable and Φ_i is obtained from $\Phi(\tau_i, a)$ in the same way as for ($Structure$).

($Subset$): $\tau = Set\_of(a_1|\tau_1, \ldots, a_n|\tau_n)$ and $a \subseteq \{a_1, \ldots, a_n\}$. Then

$$\Phi(\tau, a) = (\bigwedge_{a_i \notin a} \neg\widetilde{p}_i) \wedge (\bigwedge_{a_i \in a} \Phi_i) \wedge [\![p]\!](\neg[\![p_0]\!]\top \leftrightarrow \widetilde{p}_1 \vee \ldots \vee \widetilde{p}_n),$$

where p_0 is a new cell variable and Φ_i, p_i, $(1 \leq i \leq n)$ are the same as for ($Structure$). Here the formula $[\![p_0]\!]\top$ indicates whether the object a is empty.

($Type\ mismatch$): In all other cases $\Phi(\tau, a) = [\![p]\!]\bot$.

In all cases the associated cell variable is p.

It is easy to see that $\Phi(\tau, a)$ is satisfiable iff $a \in D_\tau$. The satisfiability algorithm transfers it into a data structure implementing the initialization

$$v := a$$

for a variable v of type τ. The interpretation p^* of the associated cell variable is an address sufficient to restore all the information about the value of v as an object of type τ. It is a natural pointer value. Specific features of the implementation are reflected in $\Phi(\tau, a)$; it plays a role of a program for building this data structure. The examples of resulting data structures are shown on Fig.1.

Note that we have chosen the variant of the program $\Phi(\tau, a)$ where all possible simplifications are already done. This job can be left to the satisfiability algorithm too. For example in the case of ($Structure$) when $\tau = \{\tau_1, \ldots, \tau_n\}$ and $a = (a_1, \ldots, a_m)$ we may take the following variant:

$$[\![p]\!](\widetilde{p}_1 \wedge \ldots \wedge \widetilde{p}_n) \wedge (\bigwedge_{i=1}^{m} \Phi_i) \wedge [\![p]\!](\widetilde{p}_1 \wedge \ldots \wedge \widetilde{p}_m).$$

It is equivalent to $\Phi(\tau, a)$ and the algorithm transfers it into the same data structure.

In order to construct a reference structure which uses only one cell instead of many containing the same record, i.e. to construct a reference structure with a functional

Structure

$\tau = \{\tau_1, \tau_2\}, \quad a = (a_1, a_2).$

$[\![p_1]\!] a_1 \wedge [\![p_2]\!] a_2 \wedge [\![p]\!] (\widetilde{p_1} \wedge \widetilde{p_2})$

$$\begin{tikzcd}[column sep=small] p: & \widetilde{p}_1 \arrow[d] & \wedge & \widetilde{p}_2 \arrow[d] \\ & \boxed{\begin{matrix} p_1 : \\ a_1 \end{matrix}} & & \boxed{\begin{matrix} p_2 : \\ a_2 \end{matrix}} \end{tikzcd}$$

Union

$\tau = \{\tau_1; \tau_2\}, \quad a \in D_{\tau_1} \cup D_{\tau_2}.$

$([\![q]\!] a_1 \vee [\![q]\!] a_2) \wedge [\![p]\!] \widetilde{q}$

$$\begin{tikzcd} p: & \widetilde{q} \arrow[d] \\ & \boxed{\begin{matrix} q : \\ a \end{matrix}} \end{tikzcd}$$

Subset

$\tau = Set\_of(a_1 : \tau_1, a_2 : \tau_2, a_3 : \tau_3), \quad a = \{a_2, a_3\}.$

$\neg \widetilde{p_1} \wedge [\![p_2]\!] a_2 \wedge [\![p_3]\!] a_3 \wedge [\![p]\!] (\neg [\![p_0]\!] \top \leftrightarrow \widetilde{p_1} \vee \widetilde{p_2} \vee \widetilde{p_3})$

$$\begin{tikzcd}[column sep=small] p: & \neg [\![p_0]\!] \top \arrow[d] & \leftrightarrow & \widetilde{p}_1 \arrow[d] & \vee & \widetilde{p}_2 \arrow[d] & \vee & \widetilde{p}_3 \arrow[d] \\ & \boxed{\begin{matrix} p_0 : \\ * \end{matrix}} & & p_1 : \Box & & \boxed{\begin{matrix} p_2 : \\ a_2 \end{matrix}} & & \boxed{\begin{matrix} p_3 : \\ a_3 \end{matrix}} \end{tikzcd}$$

$\tau = Set\_of(a_1 : \tau_1, a_2 : \tau_2, a_3 : \tau_3), \quad a = \emptyset.$

$\neg \widetilde{p_1} \wedge \neg \widetilde{p_2} \wedge \neg \widetilde{p_3} \wedge [\![p]\!] (\neg [\![p_0]\!] \top \leftrightarrow \widetilde{p_1} \vee \widetilde{p_2} \vee \widetilde{p_3})$

$$\begin{tikzcd}[column sep=small] p: & \neg [\![p_0]\!] \top \arrow[d] & \leftrightarrow & \widetilde{p}_1 \arrow[d] & \vee & \widetilde{p}_2 \arrow[d] & \vee & \widetilde{p}_3 \arrow[d] \\ & \boxed{\begin{matrix} p_0 : \\ \top \end{matrix}} & & p_1 : \Box & & p_2 : \Box & & p_3 : \Box \end{tikzcd}$$

Figure 1:

conversion of the ground storage relation $(R\theta)^{-1}$ or, even more, with invertible reading procedure θ, we introduce the logics $\mathcal{LR}_1$ and $\mathcal{LR}_{1-1}$. The logic $\mathcal{LR}_1$ is $\mathcal{LR}$ + (A4) where (A4) is the following axiom scheme:

$$\text{(A4)} \qquad [\![p]\!]A \wedge [\![p']\!]A \;\rightarrow (B \leftrightarrow B[p'/p]).$$

$\mathcal{LR}_{1-1}$ is the modification of $\mathcal{LR}$ where the "conditional" equality

$$p = q \Rightarrow \widehat{p} = \widehat{q}$$

is replaced by

$$p = q \Leftrightarrow \widehat{p} = \widehat{q}.$$

5.2 Theorem. *For any labeled modal formula F*

1. $\mathcal{LR}_1 \vdash F$ iff F^* is valid for all interpretations $*$ in finite reference structures with functional relation $(R\theta)^{-1}$;
2. $\mathcal{LR}_{1-1} \vdash A$ iff A^* is valid for all interpretations $*$ in finite reference structures with invertible reading procedure θ.

Proof. Similar to the proof of the completeness Theorem 4.2. ■

The logics $\mathcal{LR}_1$ and $\mathcal{LR}_{1-1}$ are also decidable. The satisfiability algorithms from the completeness proofs for these logics can be used in the same way as that for $\mathcal{LR}$ to construct reference structures without double stored sentences. The complexity bounds from Theorem 4.17 are also preserved.

References

[1] S. Artëmov and T. Strassen. Functionality in the Basic Logic of Proofs. Technical report IAM 93-004, Universität Bern, January, 1993.

[2] J.-L.Lasser, M.J.Maher and K.Marriot. Unification revisited. In: "Foundations of Deductive Databases and Logic Programming" J.Minker (Ed.), Morgan Kauffman, pp.587-626, 1987.

[3] S. Artëmov. Logic of proofs. *Annals of Pure and Applied Logic*, v.67, No. 1, pp. 29-59, 1994.

[4] M.S.Paterson , M.N.Wegman. Linear unification. *J.Comput.Syst.Sci.* 16, 2, 158-167, 1978

[5] A.Martelli, U.Montanary. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems* 4, 258-282, 1982.

[6] F.Baader and J.H.Siekmann. Unification Theory. In: "Handbook of Logic in Artificial Intelligence and Logic Programming" D.M.Gabbay, C.J.Hogger, and J.A.Robinson (Ed.), Oxford University Press, 1994.