Axiomatization of a Denotational Semantics for First-order Logic

C. F. M. VERMEULEN, *Philosophy Department, University of Auckland, Private Bag 92019, Auckland 1, New Zealand.*[1]
Email: k.vermeulen@auckland.ac.nz

Abstract

An axiomatization is presented of the denotational semantics for first order language of Apt [1]. The goal is to obtain a rational reconstruction of the intuitions underlying this semantics. The axiomatization combines ideas about four valued logic with facts about substitutions. Soundness and completeness of the axiomatization are established. From the completeness proof a decision procedure is obtained that shows how four valued logic and order sensitivity of substitution together add up in a natural way to the denotational semantics for the language of first order logic as proposed by Apt.

Keywords: Declarative programming, denotational semantics, first order logic, axiomatization.

1 Axiomatization as rational reconstruction

In [1] Apt provides a denotational semantics for the first order langauge as a programming language. The semantics assigns to formulas of first order logic either an error message or a denotation that can be perceived as a declarative program. This assignment of programs and error messages is *denotational*. It also respects the traditional semantics of first order logic, i.e., there is a *soundness* proof for the semantics. The standard semantics for the first order language leads to undecidability, but Apt's denotational semantics has an *error* message to avoid this lack of computational realism.

Quite a bit of further work on the proposal has already been carried out. For example, in [3] and [10], ways of extending the computation power of the initial proposal have been considered. However, in this paper we consider an axiomatization of the core of the approach: Apt's semantics as presented in [1]. The axiomatization tells us whether two programs, ϕ and ψ say, have the same denotation. As was pointed out above, the assignment of error messages is such that computational realism is ensured. This means in particular that the computation of the denotations is a decidable job. Hence the semantics itself can be seen as a decision procedure for equality and thus the search for an axiomatization is not to be confused with a quest for a decision procedure. Instead it is more appropriate to think of it as an attempt to summarize the basic principles that *underly* the semantics:

axiomatization as rational reconstruction.

The results presented below provide such a reconstruction. We present the computation mechanism as a natural combination of two ingredients. The first ingredient is the distinction of four cases, a kind of four valued logic. The second ingredient is the way substitutions are

[1]The Philosophy Department much regrets the death of 'Kees' Vermeulen, who died not long after correcting the proofs of this article.

composed to compute answers. Regarding substitutions as computed answers is standard practise in the declarative programming paradigm. The relevant details about substitutions will be given later on. The four valued logic controls the computation of answers. It makes decisions about continuation of the computation depending on the four values for the next subformula (or: subprogram):

- **1** (for *truth* or *success*): the current substitution already provides the answer for the subprogram to follow;
- **0** (for *falsity* or *failure*): the current substitution causes the next subprogram to fail;
- **e** (for *error*): the next subprogram cannot be executed and execution of the overall program should be terminated;
- $\{x/r\}$: the next subprogram achieves that the current substitution is extended with the value r for x.

Here the fourth case is, of course, not really *one* case. It is a whole family consisting of all the possible basic computation steps: the different options for extending the substitution by assigning r to x. So, *via* the 'fourth value' the two ingredients of the semantics interact. In the axiomatization the distinction of the four truth values allows us to separate axioms for the substitutions from axioms for the four valued logic. In this way it gives the rational reconstruction we are looking for.

Below we first introduce Apt's denotational semantics for first order logic. Next we list the relevant semantic facts about the substitutions that are *generated* by the semantics. Then we present the calculus, discussing its soundness as we proceed. Finally, we check the completeness of the calculus. Then we will see how the calculus can be used to provide *normal forms* for formulas and we present a procedure by which these normal forms can be obtained using the axioms of the system. Finally we summarize the main results and discuss prospects for further work.

2 A denotational semantics for first order logic

We start by recalling the semantics provided in [1]. The idea of this semantics is to give a uniform computational meaning to the first-order formulas, independent of the underlying interpretation. Of course the paper [1] itself is the best source for motivation and explanation, but we present all the crucial ingredients concisely in this section.

Definition 2.1
Assume a language of terms L and an algebra $\mathcal{J}$ for it.

- D is the domain of the algebra $\mathcal{J}$ and VAR the set of variables of the language.
- Consider a term of L in which we replace some of the variables by the elements of the domain D. We call the resulting object a *generalized term*.
- Given a generalized term t, we define its *$\mathcal{J}$-evaluation* as follows. Each ground term of L evaluates to a unique value in $\mathcal{J}$. Given a generalized term t replace each maximal ground subterm of t by its value in $\mathcal{J}$. We call the resulting generalized term a *$\mathcal{J}$-term* and denote it by $[\![t]\!]_{\mathcal{J}}$.
- By a *$\mathcal{J}$-substitution* we mean a finite mapping from variables to $\mathcal{J}$-terms which assigns to each variable x in its domain a $\mathcal{J}$-term different from x. We write $dom(\theta)$ for the

domain of θ. If $dom(\theta) = \{x_1, \ldots, x_n\}$ and (for each $1 \leq i \leq n$) the $\mathcal{J}$-term assigned to x_i is h_i, then we may write θ as $\{x_1/h_1, \ldots, x_n/h_n\}$ and $x_i\theta = h_i$.

- The *range* of θ, $range(\theta)$, is the set of values $\{x\theta : x \in dom(\theta)\}$.
- Application of a $\mathcal{J}$-substitution θ to a generalized term t can be performed in the standard way and we denote the result by $t\theta$. But $t\theta$ may not be an $\mathcal{J}$-evaluated term. So a next $\mathcal{J}$-evaluation step is required to find the $\mathcal{J}$-evaluation as $[\![t\theta]\!]_{\mathcal{J}}$.
- A *composition of two $\mathcal{J}$-substitutions θ and η*, written as $\theta\eta$, is defined as the unique $\mathcal{J}$-substitution γ such that for each variable x

$$x\gamma = [\![(x\theta)\eta]\!]_{\mathcal{J}}.$$

$\mathcal{J}$-substitutions generalize both the usual substitutions and valuations, which assign domain values to variables. This helps to stay as close as possible to the intended algebra $\mathcal{J}$ throughout the computation.

Let's briefly consider some examples to see the crucial differences between the situation here and the standard situation. Let $x \in$ VAR, a be a constant of L that denotes $\mathbf{a}$ and f a unary function symbol of L that denotes $\mathbf{f}$. Then we find that: $f(x)$, $f(a)$, $f(\mathbf{a})$ and $\mathbf{f}(\mathbf{a})$ are all generalized terms, but $\mathbf{f}(a)$ is not. As $\mathcal{J}$-values we find: $[\![f(x)]\!]_{\mathcal{J}} = f(x)$, since we cannot evaluate the term $f(x)$ any further. All other examples of generalized terms have $\mathcal{J}$-value $\mathbf{f}(\mathbf{a})$ (which is just an element of the domain D!).

For the comparison of (generalized) substitutions and $\mathcal{J}$-substitutions, consider the assignment of the term $f(a)$ to the variable $y \in$ VAR: $\{y/f(a)\}$. This is a substitution, but not an $\mathcal{J}$-substitution, as $f(a)$ is not an $\mathcal{J}$-term. $\eta = \{y/\mathbf{f}(\mathbf{a})\}$ is the corresponding $\mathcal{J}$-substitution. To compose $\mathcal{J}$-substitutions we sometimes have to do some additional evaluation. For example, if we compose η with $\theta = \{x/f(y)\}$, we first get: $\{x/f(y)\}\{y/\mathbf{f}(\mathbf{a})\} = \{x/f(\mathbf{f}(\mathbf{a})),\ y/\mathbf{f}(\mathbf{a})\}$. But this is not an $\mathcal{J}$-substitution, as $f(\mathbf{f}(\mathbf{a}))$ can be evaluated further: $[\![f(\mathbf{f}(\mathbf{a}))]\!]_{\mathcal{J}} = \mathbf{f}(\mathbf{f}(\mathbf{a})) \in D$. This, then, is the proper outcome for the composition of η and θ, considered as $\mathcal{J}$-substitutions.:

$$\theta\eta = \{x/f(y)\}\{y/\mathbf{f}(\mathbf{a})\} = \{x/[\![f(\mathbf{f}(\mathbf{a}))]\!]_{\mathcal{J}},\ y/\mathbf{f}(\mathbf{a})\} = \{x/\mathbf{f}(\mathbf{f}(\mathbf{a})),\ y/\mathbf{f}(\mathbf{a})\}.$$

After these preliminary definitions we consider the semantics of an equation between two generalized terms (so, *a fortiori*, between two terms). Here and elsewhere we do not indicate the dependency of the semantics on the underlying interpretation or algebra.

$$[\![s = t]\!](\theta) := \begin{cases} \{\theta\{s\theta/[\![t\theta]\!]_{\mathcal{J}}\}\} & \text{if } s\theta \text{ is a variable that does not occur in } t\theta, \\ \{\theta\{t\theta/[\![s\theta]\!]_{\mathcal{J}}\}\} & \text{if } t\theta \text{ is a variable that does not occur in } s\theta \\ & \text{and } s\theta \text{ is not a variable,} \\ \{\theta\} & \text{if } [\![s\theta]\!]_{\mathcal{J}} \text{ and } [\![t\theta]\!]_{\mathcal{J}} \text{ are identical,} \\ \emptyset & \text{if } s\theta \text{ and } t\theta \text{ are ground and } [\![s\theta]\!]_{\mathcal{J}} \neq [\![t\theta]\!]_{\mathcal{J}}, \\ \{error\} & \text{otherwise.} \end{cases}$$

The language may have other atomic statements, of form $p(t_1, \ldots, t_n)$ say. For them we have the following notation and terminology. Consider an interpretation $\mathcal{J}$. We denote by $p_{\mathcal{J}}$ the interpretation of p according to $\mathcal{J}$. Then:

$p(t_1, \ldots, t_n)\theta$ is:

- *true* if $p(t_1, \ldots, t_n)\theta$ is ground and $([\![t_1\theta]\!]_{\mathcal{J}}, \ldots, [\![t_n\theta]\!]_{\mathcal{J}}) \in p_{\mathcal{J}}$
- *false* if $p(t_1, \ldots, t_n)\theta$ is ground and $([\![t_1\theta]\!]_{\mathcal{J}}, \ldots, [\![t_n\theta]\!]_{\mathcal{J}}) \notin p_{\mathcal{J}}$

To deal with existential quantification we use the DROP_x operation on substitutions. It is defined (for each $x, y \in \text{VAR}$) as follows:

- $x\ \text{DROP}_x(\theta)\ =\ x$
- $y\ \text{DROP}_x(\theta)\ =\ y\theta$ for $y \neq x$

For *error* we set: $\text{DROP}_x(\mathit{error}) = \mathit{error}$. DROP_x is extended in the standard way to sets Θ that contain substitutions and possibly also *error*: $\text{DROP}_x(\Theta) = \bigcup\{\text{DROP}_x(\theta) : \theta \in \Theta\}$.

Now $[\![\cdot]\!]$ is extended to the full language of first order logic by structural induction as follows. (Here At is an atomic formula other than an equation.)

- $[\![At]\!](\theta) := \begin{cases} \{\theta\} & \text{if } At\theta \text{ is true,} \\ \emptyset & \text{if } At\theta \text{ is false,} \\ \{\mathit{error}\} & \text{otherwise, that is if } At\theta \text{ is not ground,} \end{cases}$
- $[\![\phi_1 \wedge \phi_2]\!](\theta) := [\![\phi_2]\!]([\![\phi_1]\!](\theta))$,
- $[\![\phi_1 \vee \phi_2]\!](\theta) := [\![\phi_1]\!](\theta) \cup [\![\phi_2]\!](\theta)$,
- $[\![\neg\phi]\!](\theta) := \begin{cases} \{\theta\} & \text{if } [\![\phi]\!](\theta) = \emptyset, \\ \emptyset & \text{if } \theta \in [\![\phi]\!](\theta), \\ \{\mathit{error}\} & \text{otherwise,} \end{cases}$
- $[\![\exists x\ \phi]\!](\theta) := \text{DROP}_u([\![\phi\{x/u\}]\!](\theta))$, where u is a fresh variable.

In the clause for $\wedge$ we read: $[\![\phi]\!](\Theta) = \{[\![\phi]\!](\theta) : \theta \in \Theta\})$.

Examples

The following examples show the way we interpret atomic formulas and conjunction. Assume the standard algebra for the language of arithmetic with the set of integers as domain. We write $\ldots, \mathbf{-2}, \mathbf{-1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$ for its elements. Each constant i evaluates to the element $\mathbf{i}$. We then have:

1. $[\![y = z - 1 \wedge z = x + 2]\!](\{x/\mathbf{1}\}) = [\![z = x + 2]\!](\{x/\mathbf{1}, y/z - \mathbf{1}\}) =$
   $\{\{x/\mathbf{1}, y/\mathbf{2}, z/\mathbf{3}\}\}$,
2. $[\![y = 1 \wedge z = 1 \wedge y - 1 = z - 1]\!](\epsilon) = \{\{y/\mathbf{1}, z/\mathbf{1}\}\}$,
3. $[\![y - 1 = z - 1 \wedge y = 1 \wedge z = 1]\!](\epsilon) = \{\mathit{error}\}$,
4. $[\![y = 1 \wedge z = 2 \wedge y < z]\!](\epsilon) = \{\{y/\mathbf{1}, z/\mathbf{2}\}\}$,
5. $[\![y < z \wedge y = 1 \wedge z = 2]\!](\epsilon) = \{\mathit{error}\}$,
6. $[\![x = 0 \wedge \neg(x = 1)]\!](\epsilon) = \{\{x/\mathbf{0}\}\}$,
7. $[\![\neg(x = 1) \wedge x = 0]\!](\epsilon) = \{\mathit{error}\}$.

Crucially, in this semantics conjunction is not commutative and consequently it is important in which order the formulas are processed. But in those cases where the order is as required, the outcome of the interpretation is simply the evaluation one would expect.

Special atoms

We add three special atomic formulas, $\mathbf{1}$, $\mathbf{0}$ and $\mathbf{e}$, to the language of predicate logic and specify their semantics as follows:[2]

[2]Hence, from now on $\mathbf{1}$, $\mathbf{0}$ are special formulas of the language and should not be confused with the values $\mathbf{1}$, $\mathbf{0}$ in the example above.

Definition 2.2
Let a substitution θ be given.

- $[\![\mathbf{1}]\!](\theta) := \{\theta\}$,
- $[\![\mathbf{0}]\!](\theta) := \emptyset$,
- $[\![\mathbf{e}]\!](\theta) := \{error\}$.

In addition we set: $[\![\mathbf{1}]\!](error) = [\![\mathbf{0}]\!](error) = [\![\mathbf{e}]\!](error) = \{error\}$.

These special atoms will make the calculus and its completeness proof easier to write and read. They do not add to the expressivity of the language, as we already had, for example, the following formulas with exactly the same semantic properties:

- $[\![\exists x\ \exists y\ x = y]\!](\theta) = \{\theta\}$
- $[\![\exists x\ \neg(x = x)]\!](\theta) = \emptyset$
- $[\![\exists x\ \exists y\ \neg(x = y)]\!](\theta) = \{error\}$

This means that it is an option to introduce the special atoms not as an extension of the language, but rather as shorthand for these formulas. But this would complicate the presentation later on.

2.1 *Extensions: order, recursion and constraints*

The proposal by Apt is elegant and attractive as a starting point for declarative programming. However, it raises two lines of questions. First there are concerns about the expressive power of the semantics: does the definition of the denotational semantics lead to *error* answers too often? If this is so, then the semantics is not very useful for programming purposes. Secondly, there are considerations of clarity. The regular way of reading first order logic no longer works in all cases, because we have introduced order sensitivity and *error* outcomes. It would be nice to have an easy way of distinguishing cases where these things are going to play a role from cases where they really do not matter. It is for this second line of questioning that the current paper was written. The calculus we present later on, neatly separates the two ingredients—the behavior of *error* and the order sensitivity—and there are different rules taking account of these different aspects of the system. But something more can be said here already.

It all starts with the semantics of equations. There it is stipulated how the equations *sometimes* work as value assignments and *sometimes* as tests that compare the values of terms. We also already gave some examples of the order sensitivity of conjunction that display how these two roles of equations interact. Here we continue with a more general example.

Consider six distinct variables $x, y, u, v, p, q \in \text{VAR}$ and set:

- ∘ $\rho_1 \ = \ x = y$
- ∘ $\rho_2 \ = \ (f(x) = f(y)\ \wedge\ u = v)$
- ∘ $\rho_3 \ = \ (f(x) = f(y)\ \wedge\ f(u) = f(v))\ \wedge\ p = q$

Each of the ρ_i has a core program first and then continues with some variable declaration. Here the core programs are: trivial/empty for ρ_1; $f(x) = f(y)$ for ρ_2: $f(x) = f(y)\ \wedge\ f(u) = f(v)$ in ρ_3. More significant instances of core programs are easy to come up with. Now there are several options of combining these programs by means of conjunction: $\rho_3 \wedge \rho_2 \wedge \rho_1$ and $\rho_1 \wedge \rho_2 \wedge \rho_3$ are just two examples. But the semantics is sensitive to order, so we should be careful. In order to perform the equality tests the programs call for, the semantics demands that the variables are properly declared. Because of the systematic way we have specified the programs above, it will be easy to check that there is just one order in which the semantics does not come up with an *error* message: $\rho_1 \wedge \rho_2 \wedge \rho_3$. In all other cases we get an *error* because some variable that is crucial for one of the tests does not have a value yet. Note that the restriction is *not* that all terms in the equations have to be ground: the program can proceed as long as the terms in the equation have the same $\mathcal{J}$-value.

We see from this, not only what the *rationale* for the assignment of *error* is, but also how problems can easily be avoided by a clear headed strategy of programming: declare variables before you try to execute a test. This makes it already clear how to *use* the language as a programming language. In case you get an *error* message from the semantics, it also tells you what type of mistake to look for. However, it does not help very much in detecting *where* you have made the mistake.

Of course, this is but an informal explanation by means of a typical example. We regard the axiomatization task as the proper, full version of the question. That is why we have qualified the goal of this paper as one of *rational reconstruction.*

The other line of questions regards the expressive power of the system and the possibility of extending its expressivity. Since the publication of Apt's paper [1] significant progress has been made in this respect, in particular in [10] and [3]. In [10] it is shown how the order sensitivity of conjunction can be reduced and how recursion by means of Clarke completions can be worked out. In [3] the connection with constraint programming is made in a general and powerful way, so that the approach can benefit from most of the special purpose tools available for constraint propagation. These results make the overall system very expressive and make it even more urgent to get a clear view on the details of the core system. This is what our axiomatization provides. We have to refer to the cited publications for full details, but include some brief remarks and examples here to show what has been achieved.

Conjunction and order

First of all, [10] discusses an option of working with a commutative form of conjunction. Let's use $\overline{\wedge}$ as a notation for this connective. Then we can specify the semantics of order insensitive conjunction simply as:

$$[\![\phi_1 \overline{\wedge} \phi_2]\!](\theta) \;=\; [\![\phi_1]\!]([\![\phi_2]\!](\theta)) \;\cup\; [\![\phi_2]\!]([\![\phi_1]\!](\theta))$$

This is an obvious thing to do, but it involves some effort to check that it works out as expected. In particular it has to be seen to that it does not lead to a conflict with the traditional reading of first order logic, i.e., that the *soundness* of the semantics is not spoiled. After all, in cases where no *error* messages are generated by the semantics, we want the answers that we get to agree with the standard reading of first order logic. Details are in [10].

Recursion

Already a more significant challenge for the system is the addition of some recursive power. This is the main contribution of [10]: it shows how we can use Clarke completions to include the option of definition-by-recursion in the system. Let's consider a simple example to see how it works. First we give the definitions of two recursive procedures $p_i(x, y)$ $(i = 1, 2)$. (Assume that the predicates r and s already have a suitable interpretation.)

- $\psi_1(x, y) \; = \; r(x, y) \; \vee \; \exists z \; (r(x, z) \; \wedge \; p_2(z, y))$
- $\psi_2(x, y) \; = \; s(x, y) \; \vee \; \exists z \; (s(x, z) \; \wedge \; p_1(z, y))$

The procedures p_i both perform finite alternations of r and s steps, but they start with a different step: p_1 first makes an r step and p_2 first makes and s step. The formulas ψ_i^α 'approximate' these procedures from below.

- $\psi_1(x, y)^0 = \bot$
- $\psi_2(x, y)^0 = \bot$
- $\psi_1(x, y)^1 = r(x, y)$
- $\psi_2(x, y)^1 = s(x, y)$
- $\psi_1(x, y)^2 = r(x, y) \; \vee \; \exists z \; (r(x, z) \; \wedge \; s(z, y))$
- $\psi_2(x, y)^2 = s(x, y) \; \vee \; \exists z \; (s(x, z) \; \wedge \; r(z, y))$
- $\psi_1(x, y)^3 = r(x, y) \; \vee \; \exists z \; (r(x, z) \; \wedge \; (s(z, y) \; \vee \; \exists u \; (s(z, u) \; \wedge r(u, y))))$
- $\psi_2(x, y)^3 = s(x, y) \; \vee \; \exists z \; (s(x, z) \; \wedge \; (r(z, y) \; \vee \; \exists u \; (r(z, u) \; \wedge s(u, y))))$
- $\ldots$

Note that the approximations ψ_i^j never contain recursive procedure calls. Hence the following definition makes sense:

$$[\![p_i(\mathbf{t})]\!](\theta) \; = \; \bigcup\{[\![\psi_i(\mathbf{t})^\alpha]\!](\theta) : \; \alpha \geq 0\}$$

This way of introducing recursive power to the system is very natural. But we have to check that it really works as intended. The crucial check is that this does indeed give the intended, *least fixed point* interpretation of recursive definitions. Details are in [10].

Constraints

Also the connection with constraint programming has been considered, in particular in [3]. Constraint programming is a very active branch of research. It develops special purpose algorithms that lead up to a considerable speed up of some very important programming tasks. For example, in writing programs for the evaluation of special types of equations, you may have tricks that do not make sense for all the equations of terms that can be written in the programming language. Such tricks are called *constraint propagation rules.* Then it is of great use if you can delegate the evaluation of the special equations to a separate component of the system where these special rules can be applied.

This trick has been used in logic programming to great satisfaction. In [3] it is shown that the denotational semantics of Apt [1] can also make the connection with constraint programming and shows how to do it in a very general and reliable way. The crucial result there shows that—given very modest, standard conditions on the form of the constraint propagation rules—this will lead to a natural extension of the denotational semantics that remains *sound* with respect to first order logic. Details are in [3].

These options for extending the expressivity of the semantics make it clear that Apt's proposal in [1] is worth taking seriously as the core of a programming language. So, a clear insight in this core is called for. Here we attend to this task by providing an axiomatization as a fully formal way of clarification and rational reconstruction.

2.2 Facts about substitutions

In subsequent sections there are going to be a lot of situations where we can benefit from the special properties of those substitutions that actually occur in the denotational semantics. In particular, we will usually be working only with substitutions that have the property that: $dom(\theta) \cap \mathrm{VAR}(range(\theta)) = \emptyset$. Instead of checking these properties in each case later on, we try to look at them all at once in this subsection.[3]

First we point out that not all $\mathcal{J}$-substitutions are really relevant for the denotational semantics of [1]. It is true, of course, that the definitions allow us to start off the computation with any $\mathcal{J}$-substitution θ. Thereby, in the trivial sense, each $\mathcal{J}$-substitution is part of the semantics. But we consider ϵ as the natural starting point for a computation and are really only interested in substitutions that can be reached from ϵ. These can be characterized using the notion of a generating sequence. The computations in the denotational semantics involve the *composition* of one-variable substitutions. By delaying the composition we obtain a kind of derivation history for the computation: a generating sequence. Such a derivation history makes it easier to follow what happens to the variables involved in a computation. The properties of generating sequences then help us to establish the facts about substitutions we need later.

DEFINITION 2.3
We define the notion of a *generating sequence* and assign a *domain* to each generating sequence as follows:

- ϵ is a generating sequence. $dom(\epsilon) = \emptyset$;
- if σ is a generating sequence and x a variable and r a ($\mathcal{J}$-evaluated) term such that $x \notin dom(\sigma)$ and $x \notin \mathrm{VAR}([\![r\sigma]\!]_{\mathcal{J}})$, then $\sigma\{x/[\![r\sigma]\!]_{\mathcal{J}}\}$ is a generating sequence (where $r\sigma$ denotes the application of σ-read-as-a-substitution to r). $dom(\sigma\{x/[\![r\sigma]\!]_{\mathcal{J}}\}) = dom(\sigma) \cup \{x\}$.

By 'σ-read-as-a-substitution', we mean: apply the one step substitutions that make up the generating sequence one by one. So, in case

$$\sigma = \{y_1/s_1\}\ldots\{y_n/r_n\}, r\sigma = (\ldots(r\{y/r_1\})\{y_2/r_2\})\ldots\{y_n/r_n\}).$$

The next thing we need is an operation on generating sequences that corresponds in the right way to composition of substitutions. A first candidate is: concatenation of sequences. This is an associative operation and ϵ behaves as a unit element, just like composition of substitutions. But the set of generating sequences is not closed under concatenation, because of the condition on variables. For example: both $\{x/y\}$ and $\{y/x\}$ are generating sequences, but $\{x/y\}\{y/x\}$ is not. Therefore, simple concatenation does not correspond in the right way to composition of substitutions. We need the following operation, called $\otimes$, instead:

[3]The facts we present can be formulated and proved in an even more general set up. (C.f. [8], [6], for example). In this section we ignore non-equational atoms, as they do not give rise to new substitutions.

Definition 2.4
- $\sigma \otimes \epsilon = \sigma$
- $\sigma \otimes \{x/r\} = \sigma$ in case $x \in dom(\sigma)$ or $x \in \text{VAR}(r\sigma)$, where we read $r\sigma$ as the application of σ-read-as-a-substitution to r
- $\sigma \otimes \{x/r\} = \sigma\{x/[\![r\sigma]\!]_{\mathcal{J}}\}$ in case $x \notin dom(\sigma)$ and $x \notin \text{VAR}([\![r\sigma]\!]_{\mathcal{J}})$, where $r\sigma$ denotes the application of σ-read-as-a-substitution to r
- $\sigma \otimes (\sigma'\{x/r\}) = (\sigma \otimes \sigma') \otimes \{x/r\}$

Now we can map generating sequences to $\mathcal{J}$-substitutions by the mapping $[\cdot]$:

Definition 2.5
- $[\epsilon] = \epsilon$
- $[\sigma \otimes \sigma'] = [\sigma][\sigma']$

In case $[\sigma] = \theta$, we call θ a generated substitution and σ a generating sequence for θ. We check that:

Proposition 2.6
$[\cdot]$ is an embedding of monoids.

Proof. First we have to check that $\otimes$ is well defined: given generating sequences σ, σ', we check that $\sigma \otimes \sigma'$ is a generating sequence as well, i.e., satisfies the conditions on variables from definition 3. Now: for sequences σ' of length 1 these conditions are seen to by the second and third case of Definition 2.4. For longer sequences σ' they follow by the (inductive) form of the final clause in Definition 2.4.

Then we check that the generating sequences with operation $\otimes$ form a monoid. Clearly ϵ is a unit. So, we just have to check associativity. This is obvious from the final clause of Definition 2.4. So, we have obtained a monoid of generating sequences with $\otimes$ as its 'product' and ϵ as its unit.

Finally we see in the definition of $[\cdot]$ that it respects unit and product of the monoids involved. ∎

Note that all the conditions on variables we have introduced do at least some work: not all $\mathcal{J}$-substitutions have a generating sequence. For example, there is no generating sequence for $\{x/f(y),\ y/f(z)\}$. The obvious candidate fails: $\{x/f(y)\}\{y/f(z)\}$ generates $\{x/f(f(z)),\ y/f(z)\}$. Also other attempts will fail because of the following property of generating sequences:

Lemma 2.7
Consider a generating sequence σ. Then:

$$dom([\sigma]) \cap \text{VAR}(range([\sigma])) = \emptyset$$

The proof is an easy induction.

Now there are two observations that jointly ensure that for the semantics of [1] only substitutions generated by generating sequences matter. First all generated substitutions occur in the semantics. We check that:

Proposition 2.8
- $\{[\epsilon]\} = [\![x = x]\!](\epsilon)$.
- if $[\sigma] = \theta$ occurs in $[\![\phi]\!](\epsilon)$, then $[\sigma \otimes \{x/r\}]$ is in $[\![\phi\ \wedge\ x = r]\!](\epsilon)$.

The proof is immediate, by the direct correspondence between the definitions of $\otimes$ and $[\![\cdot]\!]$. We can also check that *only* generated substitutions occur in the denotational semantics.

PROPOSITION 2.9
If $\theta \in [\![\phi]\!](\eta)$ for some formula ϕ, then θ is a generated substitution.

PROOF. The proof is an induction. For the interpretation of equations we note that the condition in the semantics of equations and the condition on generating sequences agree exactly. Disjunctions and negations do not generate new substitutions.

Conjunctions generate new substitutions by function composition: $\theta\theta'$ say. But now the property follows from an easy 'second' induction on the length of the generating sequence for θ' (which is available by induction hypothesis).

For existential formulas the argument is a bit more substantial. Consider $\text{DROP}_x(\theta)$ for a θ that is generated, say $\theta = [\sigma]$. If $x \notin dom(\sigma)$, things are easy: $\text{DROP}_x(\theta) = \theta = [\sigma]$. So, consider σ for which $x \in dom(\sigma)$. If σ is of the form $\sigma'\{x/r\}$, we can simply leave out the final $\{x/r\}$ and get σ' as a generating sequence for $\text{DROP}_x(\theta)$: $\text{DROP}_x(\theta) = [\sigma']$. Else we observe that for generating sequences of length 2:

$$[\{x/r\}\{y/s\}] = [\{y/s\{x/r\}\}\{x/r\{y/s\}\}]$$

By applying this observation repeatedly, we can always obtain a generating sequence σ for θ that is of the form $\sigma'\{x/r\}$. ∎

The fact that the semantics only produces generated substitutions is used implicitly in what is to follow. From now on we will only be considering generated substitutions. We have seen that this means in particular that for θ the $dom(\theta)$ and the $range(\theta)$ do not share variables.

The fact about switching the order in generating sequences that we saw in the DROP_x case in the proof above, will be crucial for the soundness of the system, especially where we consider the SWITCH axiom. Next we list some more observations that are crucial for the soundness of the SWITCH axioms in the calculus.

The following proposition concerns situations where $[\![s = s']\!](\theta) = \{\theta\{x/r\}\}$. We call a formula ψ deterministic if it does not contain $\vee$. $\mathbf{0}$ is used as 'shorthand' for $\emptyset$ and $\mathbf{e}$ stands for $\{error\}$.

PROPOSITION 2.10
Let ψ be a deterministic formula.

Then $[\![\psi\{s/s'\}]\!](\theta)$ and $[\![\psi]\!](\{s/s'\}\theta)$ display the following case wise correspondence:

	$[\![\psi\{s/s'\}]\!](\theta)$	$[\![\psi]\!](\{s/s'\}\theta)$
(i)	$\mathbf{0}$	$\mathbf{0}$
(ii)	$\{\theta\}$	$\{\{s/s'\}\theta\}$
(iii)	$\{\theta\theta'\}$	$\{\{s/s'\}\theta\theta'\}$
(iv)	$\mathbf{e}$	$\mathbf{e}$

PROOF. The proof is by induction on the construction of ψ.[4]

Atoms:

For an atomic formula $t = t'$ we observe that the computation of $[\![(t = t')\{s/s'\}]\!](\theta)$ is fixed by $(t\{s/s'\})\theta$ and $(t'\{s/s'\})\theta$. In exactly the same way the computation of $[\![t = t']\!](\{s/s'\}\theta)$ is

[4]We use i.h. as shorthand for induction hypothesis.

fixed by $t(\{s/s'\}\theta)$ and $t'(\{s/s'\}\theta)$. But of course: $(t\{s/s'\})\theta = t(\{s/s'\}\theta)$ and $(t'\{s/s'\})\theta = t'(\{s/s'\}\theta)$. This establishes the base case of the induction.

Negations:

We check each case. Note that case (iii) cannot occur.

(i): $[\![\neg\phi\{s/s'\}]\!](\theta) = \mathbf{0}$ iff $[\![\phi\{s/s'\}]\!](\theta) = \{\theta\}$ iff (i.h. for (ii)) $[\![\phi]\!](\{s/s'\}\theta) = \{\{s/s'\}\theta\}$ iff $[\![\neg\phi]\!](\{s/s'\}\theta) = \mathbf{0}$.

(ii): $[\![\neg\phi\{s/s'\}]\!](\theta) = \{\theta\}$ iff $[\![\phi\{s/s'\}]\!](\theta) = \mathbf{0}$ iff (i.h. for (i)) $[\![\phi]\!](\{s/s'\}\theta) = \mathbf{0}$ iff $[\![\neg\phi]\!](\{s/s'\}\theta) = \{\{s/s'\}\theta\}$.

(iv): $[\![\neg\phi\{s/s'\}]\!](\theta) = \mathbf{e}$ iff $[\![\phi\{s/s'\}]\!](\theta) = \mathbf{e}$ or $[\![\phi\{s/s'\}]\!](\theta) = \{\theta\theta'\}$ iff (i.h. applied case-by-case: cases (iii) and (iv) respectively) $[\![\phi]\!](\{s/s'\}\theta) = \mathbf{e}$ or $[\![\phi]\!](\{s/s'\}\theta) = \{\{s/s'\}\theta\theta'\}$ iff $[\![\neg\phi]\!](\{s/s'\}\theta) = \mathbf{e}$.

Existential quantification:

In this case we have to consider $(\exists y\ \phi)\{s/s'\}$ and a sufficiently fresh parameter u. Variable clashes in the substitution are taken care of in the usual way. So, we find: $(\exists y\ \phi)\{s/s'\} = \exists y\ \phi\{s/s'\}$. Now the freshness of u is enough to ensure: $\{s/s'\}\{y/u\} = \{y/u\}\{s/s'\}$. Hence we get: $[\![\exists y\ \phi\{s/s'\}]\!](\theta) = [\![\phi\{s/s'\}\{y/u\}]\!](\theta) = [\![\phi\{y/u\}\{s/s'\}]\!](\theta)$ and by i.h. $[\![\phi\{y/u\}\{s/s'\}]\!](\theta) = [\![\phi\{y/u\}]\!](\{s/s'\}\theta) = [\![\exists y\ \phi]\!](\{s/s'\}\theta)$, as required.

Conjunction:

There are 16 combinations of cases to take into account. In each case the induction goes through. As an example we look at the combination of (iii) for ϕ_1 with (iii) for ϕ_2:

$[\![\phi_1\{s/s'\}]\!](\theta) = \{\theta\theta'\}$ iff (i.h.)
$[\![\phi_1]\!](\{s/s'\}\theta) = \{\{s/s'\}\theta\theta'\}$ and
$[\![\phi_2\{s/s'\}]\!](\theta\theta') = \{\theta\theta'\theta''\}$ iff (i.h.)
$[\![\phi_2]\!](\{s/s'\}\theta\theta') = \{\{s/s'\}\theta\theta'\theta''\}$

as required. ■

In the situation of this proposition we have the following:

LEMMA 2.11
$\{s/s'\}\theta\{x/r\} = \theta\{x/r\}$, where $s\theta = x$ and $[\![s'\theta]\!]_{\mathcal{J}} = r$.

PROOF. There are three cases:

- $s\{s/s'\}\theta\{x/r\} = [\![s'\theta]\!]_{\mathcal{J}}\{x/r\} = r\{x/r\} = r = x\{x/r\} = s\theta\{x/r\}$ (using $x \notin \text{VAR}(r)$)
- $x\{s/s'\}\theta\{x/r\} = x\theta\{x/r\}$ (in case $x \neq s$)
- $y\{s/s'\}\theta\{x/r\} = y\theta\{x/r\}$ (in case $y \neq x$ and $y \neq s$)

■

This lemma can also be formulated in terms of idempotency of generated substitutions: if θ is a generated substitution, then $\theta\theta = \theta$. (Check this with an easy induction on the length of the generating sequence.) Now, combining the proposition and the lemma, we get the crucial proposition for switching formulas. (We continue the notation introduced in the lemma.)

PROPOSITION 2.12
Assume that $[\![s = s']\!](\theta) = \{\theta\{x/r\}\}$. Then:
$[\![s = s' \ \wedge \ \phi]\!](\theta) = [\![\phi\{s/s'\} \ \wedge \ s = s']\!](\theta)$.

PROOF.

$$
\begin{array}{ll}
[\![s = s' \ \wedge \ \phi]\!](\theta) = & \\
[\![\phi]\!]([\![s = s']\!](\theta)) = & (\text{by assumption}) \\
[\![\phi]\!](\theta\{x/r\}) = & (\text{by lemma 2.11}) \\
[\![\phi]\!](\{s/s'\}\theta\{x/r\}) = & (\text{by assumption}) \\
[\![s = s']\!]([\![\phi]\!](\{s/s'\}\theta)) = & (\text{by proposition 2.10}) \\
[\![s = s']\!]([\![\phi\{s/s'\}]\!](\theta)) = & \\
[\![\phi\{s/s'\} \ \wedge \ s = s']\!](\theta) &
\end{array}
$$

∎

This gives us exactly what we need to show for the soundness of the SWITCH axiom, that allows us to move ϕ to the left inside a formula at the cost of a substitution (under the conditions specified in the calculus). Here we anticipate terminology that will be introduced carefully later on, but an example can be given already:

$[\![x = f(y) \wedge f(x) = ff(y)]\!](\epsilon) \neq$
$[\![f(x) = ff(y) \wedge x = f(y)]\!](\epsilon)$, but:
$[\![x = f(y) \wedge f(x) = ff(y)]\!](\epsilon) =$
$[\![f(x) = ff(y)\{x/f(y)\} \wedge x = f(y)]\!](\epsilon)$ holds instead.

2.3 Danger sets

We have seen that we have to make sure to choose fresh variables in $[\![\exists x \ \phi]\!](\theta)$. But sometimes even fresh variables can be dangerous. We can see this from the following example: $[\![\exists x \ y = x + 1 \ \wedge \ y = 5 \ \wedge \ u = 7]\!](\epsilon)$. In the semantics of this formula we choose a fresh variable to compute $[\![\exists x \ y = x + 1]\!](\epsilon)$, but at this point both u and v are still fresh! Then these different choices of fresh variables lead to unacceptably different outcomes in the semantics: using u leads to $[\![\exists x \ y = x + 1 \ \wedge \ y = 5 \ \wedge \ u = 7]\!](\epsilon) = \emptyset$ and using v leads to $[\![\exists x \ y = x + 1 \ \wedge \ y = 5 \ \wedge \ u = 7]\!](\epsilon) = \{\{y/\mathbf{5}\}\{u/\mathbf{7}\}\}$.

To prevent this type of pathology we can extend the semantics with a *danger set*, X say. To be precise, we can define $[\![\cdot]\!]$ as a mapping on pairs $\langle\theta, X\rangle$ for a substitution θ and a set of variables $X \subseteq$ VAR. In most cases the definition of $[\![\phi]\!]$ leaves the danger set X untouched. The exception is the clause for $\exists x \ \phi$, where we now set:

$\langle\theta', X'\rangle \in [\![\exists x \ \phi]\!](\langle\theta, X\rangle) :\Leftrightarrow$
$\langle\theta'', X'\rangle \in [\![\phi\{x/u\}]\!](\langle\theta, X \cup \{u\}\rangle)$ &
$\theta' \in \text{DROP}_u(\theta'')$,
for some $u \notin X \cup dom(\theta)$

So, we make sure that u is not dangerous yet and add u to the danger set X in the subcomputation of $[\![\phi\{x/u\}]\!]$. Then we continue the computation with the extended danger set X'. Now it is easy to check that we get the intended output for $[\![\phi]\!]$ by starting with danger set $X = \text{FREEVAR}(\phi)$, the set of free variables of ϕ.

The use of danger sets and the usual notion of equality-up-to-renaming are distinct issues. We could easily standardize the choice of variables to replace the 'local' freshness requirement. But this does not eliminate the 'global' danger of the use of u in the example above.

By now we have seen the denotational semantics, some examples of how it works and we have seen the facts about the semantics that are crucial for the soundness check later on. Next we turn to the calculus itself.

3 The calculus

We present the calculus that axiomatizes the denotational semantics for the first order language.

First we have to be precise about what the calculus calculates. Basically we are interested in the question of equality of denotations: for ϕ and ψ we want to know whether $[\![\phi]\!]$ and $[\![\psi]\!]$ are equal. Stated in this way the question is hard to answer. We have to be a bit more precise. First step is to ask the question relative to a given *input* state θ. This way the equality question specializes into many cases: we consider $[\![\phi]\!](\theta)$ and $[\![\psi]\!](\theta)$ separately for each θ. Next we notice that the strict equality of these two sets of substitutions is not really what we are interested in. $[\![\exists x\ \phi]\!](\theta)$ is specified only up to the choice of a suitable variable u. The choice of u can have a lasting effect. To see this, consider the formula $\exists x\ y = f(x)$. In the interpretation both

$$[\![\exists x\ y = f(x)]\!](\epsilon) = \text{DROP}_u\{y/f_{\mathcal{J}}(u)\} = \{y/f_{\mathcal{J}}(u)\} \text{ and}$$
$$[\![\exists x\ y = f(x)]\!](\epsilon) = \text{DROP}_v\{y/f_{\mathcal{J}}(v)\} = \{y/f_{\mathcal{J}}(v)\}$$

are allowed. Note that this can only happen if the fresh u (or v) is not removed by the DROP instruction. This corresponds to substitutions θ where $u \in \mathit{range}(\theta) \backslash \mathit{dom}(\theta)$. We will call such occurrences of variables in substitutions *parametric variables.* The example shows that for the comparison of substitutions we should only consider equality *up to* choice of parametric variables. We will write $\theta \approx \eta$ for this notion of equality of substitutions.

Definition 3.1
Let substitutions θ and η be given. Then:

- $\mathit{dom}(\theta) = \{x :\ x\theta \neq x\}$
- $\mathit{range}(\theta) = \{x\theta :\ x\theta \neq x\}$
- $\mathit{parm}(\theta) = \text{VAR}(\mathit{range}(\theta)) \backslash \mathit{dom}(\theta)$
- $\theta \approx \eta$ iff $\exists\ \alpha_\theta : \text{VAR} \to \text{VAR}\ \exists\ \alpha_\eta : \text{VAR} \to \text{VAR}$: $\mathit{dom}(\alpha_\theta) \subseteq \mathit{parm}(\theta)\ \&\ \mathit{dom}(\alpha_\eta) \subseteq \mathit{parm}(\eta)\ \&$ $\theta\alpha_\theta = \eta\alpha_\eta$

For sets of substitutions Θ and H:

- $\Theta \approx H$ iff $\exists\alpha_\Theta\ \exists\alpha_H : \forall\theta \in \Theta\ \exists\eta \in H\ \theta\alpha_\Theta = \eta\alpha_H\ \&$ $\forall\eta \in H\ \exists\theta \in \Theta\ \theta\alpha_\Theta = \eta\alpha_H$

This way, to compare θ and η we use α_θ and α_η to make up for any unfortunate choices of parameters that we have made. Another way of making the relevant point is to say that parametric variables have an *existential* reading.[5]

Then the notation is extended to *sets* of substitutions and specialized to sets that occur in the semantics of [1] in the obvious way.

The calculus we present below is also *parametric* in other ways. Its parameters are: θ and Pa. θ is the input substitution: it gives the values of the variables at the start of the computation. Throughout any deduction in the calculus θ stays fixed. So, we provide a family of calculi instead of just one: one calculus for each θ. One of the calculi of particular interest is the one for $\theta = \epsilon$. Pa is the set of parametric variables that we wish to avoid while we apply the rules of the calculus. Recall that the parametric variables are those variables that have an existential reading. If we want the calculus to make sense, we better make sure that we start deductions of $\phi \approx_\theta \psi$ with a Pa-set that includes the parametric variables that occur in θ, $\text{FREEVAR}(\phi\theta)$ and $\text{FREEVAR}(\psi\theta)$ and avoid those variables in application of substitution rules and $\exists x$ axioms (also see the side conditions on those rules and axioms below). Pa can grow during a deduction in the calculus: if an application of an $\exists x$ axiom involves a new u, we add this u to the set Pa.

Finally we extend the terminology with the notion of *value* which is defined as follows:

DEFINITION 3.2

$val(s = t,\ \theta)$	
$x = r$	if $s\theta = x$, $[\![t\theta]\!]_{\mathcal{J}} = r$ and x does not occur in r
$x = r$	if $t\theta = x$, $[\![s\theta]\!]_{\mathcal{J}} = r$, x does not occur in r and $r \notin \text{VAR}$
$\mathbf{1}$	if $[\![s\theta]\!]_{\mathcal{J}} = [\![t\theta]\!]_{\mathcal{J}}$
$\mathbf{0}$	if $[\![s\theta]\!]_{\mathcal{J}}$ and $[\![t\theta]\!]_{\mathcal{J}}$ are both ground and $[\![s\theta]\!]_{\mathcal{J}} \neq [\![t\theta]\!]_{\mathcal{J}}$
$\mathbf{e}$	otherwise
$val(At,\ \theta)$	for non-equational atoms At
$\mathbf{1}$	if $At = \mathbf{1}$
$\mathbf{0}$	if $At = \mathbf{0}$
$\mathbf{e}$	if $At = \mathbf{e}$
$\mathbf{1}$	if At is $A(\mathbf{t})$ and $A(\mathbf{t})$ is ground and $[\![t\theta]\!]_{\mathcal{J}} \in A_{\mathcal{I}}$
$\mathbf{0}$	if At is $A(\mathbf{t})$ and $A(\mathbf{t})$ is ground and $[\![t\theta]\!]_{\mathcal{J}} \notin A_{\mathcal{I}}$
$\mathbf{e}$	if At is $A(\mathbf{t})$ and $A(\mathbf{t})$ is not ground

We now turn to the presentation of the axioms and rules of the calculus. We will check soundness as we move along. In the presentation we distinguish three sets:

- the fully general axioms;
- special axioms for conjunctions of atoms;
- the rules of the calculus.

General axioms

In the general axioms ϕ, ψ and χ are completely arbitrary. There are no side conditions on them.

[5]These concepts are not new in the context of declarative programming. Cf., [6] and [8], for example.

General axioms		
$At \approx_\theta val(At,\theta)$ for atomic propositions At		
$\neg\mathbf{1} \approx_\theta \mathbf{0}$	$\exists x\ \mathbf{1} \approx_\theta \mathbf{1}$	$\mathbf{1} \wedge \phi \approx_\theta \phi$
$\neg\mathbf{0} \approx_\theta \mathbf{1}$	$\exists x\ \mathbf{0} \approx_\theta \mathbf{0}$	$\mathbf{0} \wedge \phi \approx_\theta \mathbf{0}$
$\neg\mathbf{e} \approx_\theta \mathbf{e}$	$\exists x\ \mathbf{e} \approx_\theta \mathbf{e}$	$\mathbf{e} \wedge \phi \approx_\theta \mathbf{e}$
$\phi \vee \mathbf{0} \approx_\theta \phi$		
$\phi \vee \phi \approx_\theta \phi$		
$\phi \vee \psi \approx_\theta \psi \vee \phi$		
$\phi \vee (\psi \vee \chi) \approx_\theta (\phi \vee \psi) \vee \chi$		
$\phi \wedge (\psi \wedge \chi) \approx_\theta (\phi \wedge \psi) \wedge \chi$		
$\phi \wedge (\psi \vee \chi) \approx_\theta (\phi \wedge \psi) \vee (\phi \wedge \chi)$		
$(\phi \vee \psi) \wedge \chi \approx_\theta (\phi \wedge \chi) \vee (\psi \wedge \chi)$		
$\exists x\ (\phi \vee \psi) \approx_\theta \exists x\ \phi\ \vee \exists x\ \psi$		

The first axiom concerns atomic formulas At. According to the axiom they are equal to their value, $val(At,\theta)$.

It is important that *val* is a purely syntactic notion on equations and special atoms. So, we are not smuggling semantics into the axiom system. At the same time, the definition of *val* simply mirrors the interpretation of equations from [1].

Non-equational axioms do spoil this picture. Here we have chosen to smuggle in a bit of 'semantics.' This seems to be the sensible thing to do given their role as built-ins. They are not syntactic but not to be covered but *our* semantics either: their semantics is built-in.

All other general axioms are obvious and their soundness is easy to check. For example, both $\vee$ and $\wedge$ are associative, as the corresponding operations are union of sets and composition of functions, respectively. Note that the general axioms tell us all we need to know about the underlying four valued logic.

There are additional axioms for conjunction that take care all we have to know about substitutions, a different part of the system that deserves a separate paragraph.

Axioms for easy conjunctions

As announced, the axioms for conjunctions take care of all the details concerning the substitutions generated by the system. It turns out that the axioms for conjunctions are a bit special: we can restrict them to the special cases of conjunctions that we call *easy*.

Easy conjunctions are conjunctions that generate substitutions that are insensitive to order. Order sensitivity depends on input assignment θ. So, we should say 'easy for θ.' A small detail about the definition of easy conjunctions is whether it is technically more convenient to allow the special atoms or not. One option is to allow atoms **1**, **0**, **e** to occur in easy conjunctions and have a non-sharing condition on the remaining conjuncts; the other option is to take care of the three special truth values elsewhere and to restrict the notion of easy conjunction to situations where all equations $s = s'$ correspond to substitution instructions: $[\![s = s']\!](\theta) = \{x/r\}$. We will follow option two.

Note that as a consequence an easy conjunction $\mathbf{At}$ will in fact always be a conjunction of equations: non-equational atoms do not have values of type $x = r$ and we have excluded special propositions as well. In what follows the condition $val(s = s', \theta) = x = r$ summarizes two cases: both $s\theta = x$ and $s'\theta = x$ are covered.

Definition 3.3
Let a conjunction of atomic formulas $\mathbf{At} = At_1 \wedge \ldots \wedge At_n$ $(n \geq 1)$ be given. We call $\mathbf{At}$ an easy conjunction (given θ) if $val(At_i, \theta) = x_i = r_i$ (all $1 \leq i \leq n$) and it does not occur that x_i occurs in r_j (any $1 \leq i, j \leq n$).

We introduce a notation convention here that we will stick to throughout the rest of the paper: using bold face notation for (finite) conjunctions.

The first axiom below is crucial for the comparison of easy conjunctions. Recall that by definition the easy conjunctions will consist of equations $s_i = s'_i$ that interpret as $\{x_i/r_i\}$ in such a way that all the substitution instructions are order independent. This makes checking $[\![\mathbf{At}]\!](\theta) \approx [\![\mathbf{At}']\!](\theta)$ an easy task. The last rule, for negation of easy conjunctions, simply follows the clause for interpretation of $\neg$: as easy conjunctions generate substitutions, the negation of easy conjunctions lead to an *error* report in the interpretation and we find $\mathbf{e}$ in the calculus here.

Axioms for easy conjunctions
$\mathbf{At} \approx_\theta \mathbf{At}'$ if both conjunctions of equations are easy and $[\![\mathbf{At}]\!](\theta) \approx [\![\mathbf{At}']\!](\theta)$
$\exists x\ (\mathbf{At} \wedge s = s') \approx_\theta \mathbf{At}\{x/u\}$ if $(\mathbf{At} \wedge s = s')$ is an easy conjunction and $val(s = s', \theta) =\ x = r$. u does not occur in FREEVAR$(\exists x\ \mathbf{At}) \cup Pa$ so far. Now u is added to Pa.
$\exists x\ \mathbf{At} \approx_\theta \mathbf{At}\{x/u\}$ if $\mathbf{At}$ is an easy conjunction and for no conjunct At and no term r: $val(At, \theta) =\ x = r$; u does not occur in FREEVAR$(\exists x\ \mathbf{At}) \cup Pa$ so far. Now u is added to Pa.
$(\mathbf{At} \wedge s_1 = s_2 \wedge \phi) \approx_\theta \mathbf{At} \wedge \phi\{s_i/s_j\} \wedge s_1 = s_2$ if $(\mathbf{At} \wedge s_1 = s_2)$ is an easy conjunction and $val(s_1 = s_2,\ \theta) = x = r,\ \ (s_i\theta = x)$
$\neg\mathbf{At} \approx_\theta \mathbf{e}$ if $\mathbf{At}$ is an easy conjunction

The axiom for $\mathbf{At} \wedge s_1 = s_2 \wedge \phi$ is called the SWITCHING axiom. It allows us to SWITCH ϕ and $s_1 = s_2$ even when order insensitivity is not (yet) ensured. But this SWITCH is only allowed at the cost of a substitution to make sure that the semantic impact of $s = s'$ on ϕ is accounted for. In the section on semantic facts we have seen a proposition that is just what is required to guarantee the soundness of this axiom. The purpose of the SWITCH axiom is threefold: first we can switch equations with a value of type $\{x/r\}$ to the right frontier of the formula. In that position it is easy to see what the effect of DROP$_x$ on the easy conjunction will be. Secondly, it allows us to move 'other stuff' to the left frontier of the formula, where it can be dealt with, for example by the axioms for the special atoms from the previous subsection. Finally the application of the SWITCHING axiom gives us the power to make conjunctions easier. The substitution instruction generated by $s = s'$ now already has been carried out in ϕ and hence fewer variables are shared. This way we can keep SWITCHING until a conjunction has become easy.

In the second and third case we consider easy conjunctions in the scope of an existential quantifier $\exists x$. In these cases the semantics will give a DROP instruction. In the calculus we check whether $\{x/r\}$ is somehow part of the effect of the conjunction. If not, we need not worry about DROPPING things and can continue with $\mathbf{At}\{x/u\}$ (where u is the parametric variable we are using for x). This is the third axiom above. We are careful about the choice of u and add it to the Pa set.

In case x does indeed appear in the domain of the substitution we get in interpreting $\mathbf{At}$, we perform two steps: first we take care that the crucial equation $s = s'$ is on the right frontier of $\mathbf{At}$. We mentioned above that we can use the SWITCH axiom to do this. Then we can simply forget the occurrence of this equation altogether: its semantic contribution has been secured by the SWITCHING and the DROP instruction says that it will have no semantic effect outside the scope of $\exists x$. This way the two rules for $\exists x\ \mathbf{At}$ cover all cases.

The rules of the calculus

Inevitably the calculus also contains some rules. Here we use equality rules and a substitution rule. The equality rules are obvious. In the substitution rule we impose a condition: the locus of the substitution should be an initial position in ψ. The proper definition of initial position makes sure that at this point it is still the input substitution θ that is relevant for the computation. Later on θ may have evolved into some 'larger' θ', but in the initial position it is still θ that matters. For the definition of initial positions we introduce the notion of a position in a formula. Then we define which positions are initial.

DEFINITION 3.4

Add ℓ as a new atomic proposition to a language of predicate logic. ℓ is called *position* (or *location*). Let ℓ occur exactly once in the formula ψ and let $\phi = \psi\{\ell/\chi\}$. Then we say that χ occurs at position (or location) ℓ in ϕ. By abuse of language we sometimes say that ℓ is a position (or location) in ϕ.

Note that using different formulas ψ of the extended language allows us to distinguish different occurrences of sub-formulas of ϕ.

DEFINITION 3.5

Consider a language of predicate logic extended with the atomic proposition ℓ. Let ℓ occur in ψ exactly once. Then we say that ℓ occurs in initial position in ψ iff one of the following conditions holds:

$\psi = \ell$
$\psi = (\psi_1 \wedge \psi_2)$ and ℓ occurs in initial position in ψ_1
$\psi = \exists x\ \psi_1$ and ℓ occurs in initial position in ψ_1
$\psi = \neg\ \psi_1$ and ℓ occurs in initial position in ψ_1

Note that disjunctions or formulas containing disjunctions do not have an initial position. We continue to call formulas without disjunctions *deterministic*.

To apply the notion of an initial position to disjunctions seems incompatible with the parallel behavior of the semantics of disjunctions. If we insist on using the concept of initial positions for disjunctions anyway, then one way would be to regard the initial position of any of the disjuncts as initial to the disjunction as a whole.

Now we can see that if ϕ and ϕ' are in initial position, substitution is sound. At such a position the input state is still θ and if we have established that $\phi \approx_\theta \phi'$, we know that they will produce equal output as well.

This explains the first half of the following table.

Rules		
$\dfrac{}{\phi \approx_\theta \phi}$	$\dfrac{\phi \approx_\theta \psi}{\psi \approx_\theta \phi}$	$\dfrac{\phi \approx_\theta \psi \qquad \psi \approx_\theta \chi}{\phi \approx_\theta \chi}$
$\dfrac{\phi \approx_\theta \phi'}{\psi\{\ell/\phi\} \approx_\theta \psi\{\ell/\phi'\}}$ provided ℓ is an initial position and ψ does not contain any of the variables in Pa.		
$\dfrac{\phi \approx_\theta \phi' \qquad \psi \approx_\theta \psi'}{\phi \vee \psi \approx_\theta \phi' \vee \psi'}$		
$\dfrac{\neg\phi \approx_\theta \mathbf{e}}{\neg(\phi \vee \psi) \approx_\theta \neg\psi \wedge \neg\phi}$		
$\dfrac{\neg\psi \approx_\theta \mathbf{e}}{\neg(\phi \vee \psi) \approx_\theta \neg\phi \wedge \neg\psi}$		
$\dfrac{\neg\phi \not\approx_\theta \mathbf{e} \qquad \neg\psi \not\approx_\theta \mathbf{e}}{\neg(\phi \vee \psi) \approx_\theta \neg\phi \wedge \neg\psi}$		

In the lower half of the table there are rules for the interaction of $\neg$ and $\vee$. These involve a switch from the order insensitive world of $\vee$ to the order sensitive world of $\wedge$. Considering all subtleties that can arise, results in the end in a case distinction depending on the values of $\neg\phi$ and $\neg\psi$. In case these involve $\mathbf{0}$ or $\mathbf{e}$, the fact that $\mathbf{0} \wedge \mathbf{e} \not\approx_\theta \mathbf{e} \wedge \mathbf{0}$ can be used as a crucial test for the order of the (de Morgan like) formula that we find in the conclusions of the rules. Note that the final rule seems to involve $\not\approx_\theta$, but this is just a way of summarizing four cases: $\neg\phi \approx_\theta \mathbf{1}/\mathbf{0}$ and $\neg\psi \approx_\theta \mathbf{1}/\mathbf{0}$.

This completes our discussion of the axioms and rules of the calculus and their soundness.

4 Completeness

In this section we present the argument for the completeness of the calculus. So, we show that, for any two given formulas ϕ and ψ and any generated substitution θ, the calculus we have presented can prove $\phi \approx_\theta \psi$ whenever $[\![\phi]\!](\theta) \approx [\![\psi]\!](\theta)$. The main line of the argument is to reduce the whole question to an application of the axiom for equality of easy equations. This reduction works along two lines: we move disjunctions from the inside of the formula out. For this we can rely on the rules and axioms that involve the interaction of $\vee$ with the other connectives. Then we obtain a disjunction of deterministic formulas which we can replace either by a special proposition: $\mathbf{1}$, $\mathbf{0}$ or $\mathbf{e}$, or else by an easy conjunction.

Another way of saying this is that we reduce all formulas ϕ to *normal forms* that are (disjunctions of) special values or easy conjunctions of equations. The normal forms are only specified up to choice of parameters. Then the equality (up to choice of parameters) of the normal forms for ϕ and ψ is clearly a question the calculus can handle, using the axiom for equality of easy conjunctions of equalities.

At the end of the section we also present the production of the normal forms in an algorithmic format. Then we use the axioms and rules as rewrite rules: $\phi \approx_\theta \phi'$ allows us to rewrite ϕ into ϕ' (or the other way around). This algorithmic style of presentation enhances the readability and the generality of the procedure. It also indicates clearly one way of actually *deciding* the equality question: a procedural answer to the rational reconstruction task that we have set for ourselves.

Disjunctions

We consider disjunctions first. It will be shown that it is possible to obtain a kind of *disjunctive normal form*: we replace ϕ with a disjunction of deterministic formulas. Then, in the next subsection we can restrict our attention to deterministic formulas.

PROPOSITION 4.1
For all ϕ there is a ϕ' such that $\phi \approx_\theta \phi'$ and in ϕ' no disjunction appears in the scope of $\wedge$, $\exists x$ or $\neg$.

PROOF. Consider a formula which contains a disjunction that is not the main connective. This disjunction occurs in one of the following configurations:

- $\exists x\ (\phi \vee \psi)$. In this case the axioms for $\exists/\vee$ interaction give: $\exists x \phi\ \vee \exists x\ \psi \approx_\theta \exists x\ (\phi \vee \psi)$. (Recall that it is equality up to parameters that we are after.)
- $\exists x\ (\phi \vee \psi)$. In this case we find: $(\phi \wedge \psi) \vee (\phi \wedge \chi) \approx_\theta \exists x\ (\phi \vee \psi)$.
- $(\phi \vee \psi) \wedge \chi$. Now we find: $(\phi \wedge \chi) \vee (\psi \wedge \chi) \approx_\theta (\phi \vee \psi) \wedge \chi$.
- $\neg(\phi \vee \psi)$. Now we have to establish first which of the rules for $\neg$ and $\vee$ interaction is appropriate.

In each case we can use some axiom or rule from the calculus to replace an occurrence of $\vee$ with occurrences of $\vee$ that are less 'deep.' This makes the recursive call in the final step harmless: the recursion is well founded. ∎

Two remarks are in order. First, we can choose to regard the situation $\phi \vee \mathbf{0}$ as a separate case, since we have a rule for dealing with that as well. But this is not strictly speaking necessary. Secondly, the rules for the interaction of $\neg$ and $\vee$ require that we check other equalities first. This may involve facts about deterministic formulas that are discussed in the next subsection. But the way we deal with deterministic formulas will make clear that the interaction is harmless.

Deterministic formulas

In the previous subsection we have seen how any formula can be replaced by a disjunction of deterministic formulas by a series of applications of the axioms and rules of the calculus. In this subsection we attend to the deterministic formulas. The overall goal is to replace them with a formula of one of four forms: $\mathbf{1}$, $\mathbf{0}$, $\mathbf{e}$ or an easy conjunction of equations, $\mathbf{s} = \mathbf{s}'$. These four forms are the *normal forms* for deterministic formulas. The equality of the normal forms can be dealt with by the calculus:

PROPOSITION 4.2
Let two normal forms ϕ and ψ be given such that $[\![\phi]\!](\theta) \approx [\![\psi]\!](\theta)$. Then $\phi \approx_\theta \psi$.

PROOF. There are axioms for the special propositions **1**, **0** and **e** and for the comparison of the other normal forms, of type **s= s'**, we have a separate axiom.[6] ∎

[6] Please recall we denote finite conjunctions by bold face. Hence **s=s'** stands for a finite conjunction $s_1 = s'_1 \wedge \ldots \wedge s_n = s'_n$.

So, the normal forms are the easy case and we would like the calculus to help us find a normal form for each formula.

PROPOSITION 4.3
For each ϕ there is a normal form ϕ' such that $\phi \approx_\theta \phi'$ and $[\![\phi]\!](\theta) \approx [\![\phi']\!](\theta)$.

At the next level of complexity we find cases where $\phi \equiv \exists x\ \psi$ or $\phi \equiv \neg\psi$ and where ψ already is a normal form. For these cases the rules for easy conjunctions can remove the connective. E.g. $\exists x\ (y = f(z) \wedge x = f(y)) \approx_\theta y = f(z)$ and $\neg(y = f(z) \wedge x = f(y)) \approx_\theta \mathbf{e}$. Then the substitution rule allows us to do this also inside a complex formula, but only if ϕ occurs in initial position.

In a formula there can be several sub-formulas in initial position. For example, in $\exists x\ (\exists v\ (x = y\ \wedge\ u = v))$, the complex formula $\exists v\ (x = y\ \wedge\ u = v)$ is in initial position, as well as the equation $x = y$. But it is clear that there always is only one *atomic* formula that is in initial position.

In case ϕ is in initial position and is of the form $\exists x\ \psi$ or $\neg\psi$ considered above, then this atom will be the first atom of ψ and we can apply the substitution of ϕ as indicated, bringing us one step closer to a normal form. But there are also cases where more work is required: the atom in initial position is not the first atom of such a ψ. We will use this observation in the proof of the proposition.

PROOF. Consider a formula ϕ that is not in normal form. We have already considered the case this formula of the form $\exists x\ \psi$ for ψ in normal form. The remaining cases are: (i) the initial atom is a special formula and hence the rules for special atoms can be applied; (ii) the initial atom is an equation. We consider this remaining case, i.e., assume that the initial atom is an equation in a configuration:

$$\mathbf{At} \wedge \chi$$

where $\mathbf{At}$ is already an easy equation, but $\mathbf{At} \wedge \chi$ is not. Then the SWITCH axiom can be applied several times until we find:

$$\mathbf{At} \wedge \chi \ \approx_\theta\ \chi\eta \wedge \mathbf{At}$$

for some suitable η. The sensible application of this fact is the one where the easy conjunction $\mathbf{At}$ is of maximal length.

Now there is a new initial atom, deeper in the formula. So, we can repeat the process for this new initial atom without danger of circularity: there is a recursive call here but the recursion is well founded. Hence after a finite amount of re-call we end up with a normal form ϕ' such that $\phi \approx_\theta \phi'$. Then the soundness of the calculus ensures that $[\![\phi]\!](\theta) \approx [\![\phi']\!](\theta)$. ■

The proposition about disjunction and these two propositions about deterministic formulas jointly establish the completeness of the calculus.

PROPOSITION 4.4 (Completeness)
Consider two formulas ϕ and ψ. Then:

if $[\![\phi]\!](\theta) \approx [\![\psi]\!](\theta)$, then $\phi \approx_\theta \psi$

PROOF. By the result on disjunction and the rules for disjunction and substitution it suffices to consider the case where both ϕ and ψ are deterministic. In this case we can find ϕ' and ψ' in normal form such that: $\phi \approx_\theta \phi'$ and $\psi \approx_\theta \psi'$. By the soundness of the calculus we may conclude that: $[\![\phi']\!](\theta) \approx [\![\psi']\!](\theta)$. But the equality of normal forms can be established in the calculus: $\phi' \approx_\theta \psi'$. Now the rules for equations can be applied to establish $\phi \approx_\theta \psi$, as required. ∎

The completeness result shows that the calculus presented captures the relevant notion of equality for the denotational semantics. Recall that the relevant notion of equality depends on the input substitution θ and allows for variation in the choice of parameters. As the axioms of the calculus clearly reflect the distinction between the element of four-valued-ness of the semantics and the element of order sensitivity, this means that we have a first answer to our quest for a rational reconstruction. This first answer is *declarative* in flavor: it is a list of axioms. In the next section we sketch how to transform this into a more *procedural* style.

4.1 A procedural reconstruction

We can present each of the cases of the completeness proof in the form of a rewrite rule. This will then give us an algorithm to decide equality in the denotational semantics. As the calculus distinguishes in a clear way the two components of the system—four values and order dependence–this will give a procedural form for rational reconstruction. Let's see how this works.

We use the following terminology. We want to distinguish situations where the 'next' connective to be considered is a conjunction from situations where the 'next' connective is a negation or an existential quantifier. For the second case we introduce the terminology: $\neg\exists$ context. We say that $x = y$ is in a $\neg\exists$ context in the formulas $\exists x \neg\ x = y$ and $\neg\exists x\ x = y$, but not in $\exists x \neg\ (\phi \wedge\ x = y\ \wedge \phi')$. As a borderline case we have an isolated atom *At*. In this case we say that *At* is in a $\neg\exists$ context, but the context is called *trivial.* The terminology is extended to conjunctions of atomic formulas **At** in the obvious way.

Now we use this terminology in the following procedure. The procedure starts from SCRATCH:

SCRATCH: consider the (maximal length) easy conjunction that starts with the atom in initial position. If this conjunction is in a $\neg\exists$ context then the routine 1 for REDUCTION applies. Else this conjunction is not in a $\neg\exists$ context, i.e., there is a next conjunct to be considered. In this case routine 2 for SWITCHING applies.

ROUTINE 1: REDUCTION:

- If the innermost connective of the $\exists\neg$ context is $\exists$, we apply the $\exists$ axiom for easy conjunctions.
- If the innermost connective of the $\exists\neg$ context is $\neg$, then we apply the $\neg$ axiom for easy conjunctions.
- If the $\exists\neg$ context is trivial, we are done.

ROUTINE 2: SWITCHING:

we are now dealing with cases where the maximal length easy conjunction that starts in initial position is not in a $\neg\exists$ context. So, there is a next conjunct to be considered.

We use the SWITCH AXIOM (and substitution rule) to switch this conjunct into initial position.

- Once it turns up in initial position, this conjunct can turn out to be an equation with value of type $\{x/r\}$: the switching rule will have seen to it that we are now dealing with an easy conjunction of longer length. We continue from SCRATCH.
- Once it has turned up in initial position, the next conjunct might turn out to be an atom with a special value: then we apply the appropriate trick for treating this special value: remove **1** or reduce in the other cases, as indicated above. We start from SCRATCH again.
- The conjunct is not atomic. Then, after switching it into initial position, we start from SCRATCH with an entirely new conjunction in initial position.

This summarizes the normalization procedure outlined before.

The algorithmic presentation makes it clear that the normalization method is a decision method as well. Of course, from the definition of the semantics it was already clear that the system is decidable, but now we have added a specific algorithm for reaching a decision about $\phi \approx_\theta \psi$. This algorithm is strongly based on the distinction between the four values for atomic formulas and the additional semantic facts about the order-sensitivity of composition of (generated) substitutions that we have discussed. The availability of such a decision method thus provides a procedural form of rational re-construction of the semantics.

5 Conclusion

In this paper we have been looking at the denotational semantics for first order language presented by Apt in [1]. Apt argues that this semantics can be seen as a natural attempt to achieve the goals of declarative programming: using logic as a programming language. This is realized in Logic Programming (LP) too, but the LP-approach to declarative programming has some peculiarities, especially in the way it treats negations—see [2], which shows, roughly speaking, that LP cannot deal with negation in a sufficiently natural way—and equations—these are always interpreted as *term* equations rather than *real* equations over a specific structure. Apt's proposal is an attempt to avoid these peculiarities. As was already pointed out in [1] this attempt is only partially successful. Some examples with negation that are not easy for LP, are handled naturally in [1] and also the treatment of = is less syntactic and more real-life. But at the same time there are examples where LP simply does better: it comes up with a natural solution where the semantics of [1] gives and *error* message.

The presentation in [1] induces two questions about the denotational semantics. The first question is about the power of the system read-as-a-programming-language: how often will an *error*-message turn up? This question was considered with considerable success elsewhere. Here these results have been summarized: limitations of the computation power of the semantics can be dealt with in extensions that allow for recursion, remove some of the order sensitivity and establish connections with constraint programming techniques. This positive news makes the next question even more relevant.

This is the question for an easy way to read the semantics. We have called this the quest for a *rational reconstruction.* This question has an informal answer: the denotational semantics of Apt's [1] simply is a combination of four valued logic and composition of substitutions. But a more formal answer is preferred. In this paper we give a proof calculus and a decision procedure for the semantics of [1] to serve that purpose. The results involve the use of normal forms and rely on facts about the ordering of equations. These facts are summarized in the SWITCH proposition and are put to use in the form of switch axioms of the calculus and switch instructions in the decision procedure.

This shows more clearly *how* in [1] Apt indeed succeeds in using first order logic as a programming language.

Further work immediately connected to this paper should involve a more precise estimate of the complexity of the decision question. Also more concise formulations of the calculus are part of the agenda. Looking at things in a somewhat wider setting, we note that the extensions of the denotational semantics that have been discussed raise the point of the extension of the calculus to a calculus for the extensions. Here the question sometimes has an easy answer: some of the extensions of computational power come with translations back into the original semantics. But especially the interaction of the constraint component and the basic system in [3] poses an interesting challenge, for the axiomatization and the complexity question both.

Acknowledgements

The research was sponsored by the Dutch organization of Scientific Research (NWO) and was carried out at the Center for Mathematics and Computer Science (CWI) in Amsterdam. I would like to thank K. Apt: he suggested the topic, pointed me to the PhD thesis of S. Seres [9] and provided useful comments on earlier drafts.

References

[1] K. R. Apt. A Denotational Semantics for First-order Logic. In *Proc. of the computational logic conference (CL2000)*, Lecture Notes in Artificial Intelligence 1861, pages 53–69. Springer Verlag, 2000.

[2] K. R. Apt and R. Bol. Logic Programming and Negation: a Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[3] K. R. Apt and C. Vermeulen. First-order Logic as a Constraint Programming Language. In A. Voronkov and M. Baaz, editors, *Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Artificial Intelligence 2514, pages 19–35, 2002.

[4] N. Dershowitz and J.-P. Jouannad. Rewrite Systems. In *Handbook of Computer Science*, (volume B), pages 243–320, 1990.

[5] F. Kamareddine and R. Nederpelt. On Stepwise Explicit Substitution. *Journal of Foundations of Computer Science*, pages 197–240, 1993.

[6] J.-L. Lassez, M.J. Maher and K. Marriott. Unification Revisited. In *Foundations of Deductive Databases and Logic Programming*, pages 587-625, 1988.

[7] J. Lloyd. *Foundations of Logic Programming* (second edition). Springer Verlag, 1987.

[8] M.J. Maher. *On Parametrized Substitutions.* IBM Research Report RC 16042, 1990.

[9] S. Seres. *The Algebra of Logic Programming*, PhD Thesis, Oxford, 2001.

[10] C. Vermeulen. More Computation Power for a Denotational Semantics for First Order Logic. In M. Baaz and J.A. Makowsky, editors, *Computer Science Logic*, Springer Lecture Notes in Computer Science 2803, pages 530-543, 2003.

Received 12 December 2003, revised June 2004.