Centrum voor Wiskunde en Informatica
**REPORT***RAPPORT*

# Theorem Proving and Programming with Dynamic First Order Logic

D.J.N. van Eijck, J.M. Heguiabehere, B. Ò Nualláin

Information Systems (INS)

**INS-R0020 October 31, 2000**

Report INS-R0020
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

Copyright © Stichting Mathematisch Centrum
P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

# Theorem Proving and Programming with Dynamic First Order Logic

Jan van Eijck
*CWI, jve@cwi.nl*

Juan Heguiabehere
*ILLC, juan@wins.uva.nl*

Breanndán Ó Nualláin
*ILLC, bon@wins.uva.nl*

*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
*and*
*ILLC, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands*

ABSTRACT

Dynamic First Order Logic results from interpreting quantification over a variable $v$ as change of valuation over the $v$ position, conjunction as sequential composition, disjunction as nondeterministic choice, and negation as (negated) test for continuation. We present a tableau style calculus for DFOL with explicit (simultaneous) substitution, prove its soundness and completeness, and point out its relevance for programming with dynamic first order logic, for automatic program analysis, and for semantics of natural language. Next, we extend this to an infinitary calculus for DFOL with iteration.

*2000 Mathematics Subject Classification:* 68T15, 68Q60, 03B35, 03B65, 03B70
*1998 ACM Computing Classification System:* D.2.4, F.3.1, F.4.1, I.2.3, I.2.7
*Keywords and Phrases:* Theorem proving, tableau calculus, dynamic logic, program verification, logic of natural language
*Note:* The research reported here was supported by Spinoza Logic in Action

## 1. Introduction

The language we use and analyse in this paper consists of formulas that can both be used for programming and for making assertions about programs. The only difference between a program and an assertion is that an assertion is a program with its computational effect blocked off. In the notation we will introduce below: if $\phi$ is a program, then $\{\phi\}$ is the assertion that the program $\phi$ can be executed. Execution of $\phi$ will in general lead to a set of computed answer substitutions, execution of $\{\phi\}$ to a yes/no answer indicating success or failure of $\phi$.

Since the formulas of our language, dynamic first order logic, can be used for description and computation alike, our calculus is both an execution mechanism for dynamic first order logic and a tool for theorem proving with dynamic first order logic. One of the benefits

of mixing calculation and assertion is that the calculus can be put to use to automatically derive assertions about programs for purposes of verification. And since DFOL has its roots in Natural Language processing (just like Prolog), we also see a future for our tool-set in a computational semantics of natural language.

We start our enterprise by developing a theory of substitution for dynamic first order logic that we then put to use in a calculus for dynamic first order logic with explicit substitution. The explicit substitutions represent the intermediate results of calculation that get carried along in the computation process. We illustrate with examples from standard first order reasoning, natural language processing, imperative programming, and derivation of postconditions for imperative programs. Finally, we develop an infinitary calculus for dynamic first order logic plus iteration, with a completeness proof. Our infinitary calculus is both more powerful than Hoare logic and more practical than existing infinitary calculi for quantified dynamic logic.

The calculus that is the subject of this paper is the computation and inference engine of a toy programming language for theorem proving and computing with dynamic first order logic, *Dynamo*.

## 2. Dynamic First Order Logic

Dynamic First Order Logic results from interpreting quantification over $v$ as change of valuation over the $v$ position, conjunction as sequential composition, disjunction as nondeterministic choice, and negation as (negated) test for continuation. See Groenendijk and Stokhof [GS91] for a presentation and Visser [Vis98] for an in-depth analysis. A sound and complete sequent style calculus for DFOL (without choice) was presented in Van Eijck [Eij99]. In this paper we present a calculus that also covers the choice operator, and that is much closer to standard analytic tableau style reasoning for FOL (see Smullyan [Smu68] for a classical presentation, and Fitting [Fit90] for connections with automated theorem proving).

For applications of DFOL to programming, the presence of the choice operation $\cup$ in the language is crucial: choice is the basis of 'if then else', and of all nondeterministic programming constructs for exploring various avenues towards a solution. It can (and has been) argued that the full expressive power of $\cup$ is not necessary for applications of DFOL to natural language semantics. In fact, the presentation of dynamic predicate logic (DPL) in [GS91] does not cover $\cup$: in DPL, choice is handled in terms of negation and conjunction, with the argument that natural language 'or' is externally static. This means that an 'or' construction behaves like a test. The present calculus deals with DFOL including choice.

A very convenient extension that we immediately add to DFOL is representation of simultaneous substitution. It is well known that substitutions are definable in DFOL. Still we will consider them as operators in their own right, in the spirit of Venema [Ven95], where substitutions are studied as modal operators. Simultaneous substitutions can in general not be expressed in terms of single substitutions without introducing auxiliary variables. E.g., the swap of variables $x$ and $y$ in the simultaneous substitution $[y/x, x/y]$ can only be expressed as a sequence of single substitutions at the expense of availing ourselves of an extra variable $z$, as $z := x; x := y; y := z$. The dynamic effect of this sequence of single substitutions is not quite the same as that of $[y/x, x/y]$, for $z := x; x := y; y := z$ changes value of $z$, while $[y/x, x/y]$ does not, and the semantics of DFOL is sensitive to such subtle differences.

Let a signature for FOL be given. We will assume the presence of individual constants,

function constants and predicate constants. Let $V$ be the set of variables, $F$ the set of function symbols, and $a : F \to \mathbb{N}$ a function that assigns to every function symbol its arity. The function symbols with arity 0 are the individual constants. The set $T$ of terms of the language is given in the familiar way, by $t ::= v \mid ft_1 \cdots t_n$, where $v$ ranges over the set of variables $V$ and $f$ over $F$, with $a(f) = n$. The subterms of a term are given as usual. Subterms need not be proper: we take every term to be a subterm of itself.

A substitution $\sigma$ is a function $V \to T$ that makes only a finite number of changes, i.e., $\sigma$ has the property that $dom(\sigma) = \{v \in V \mid \sigma(v) \neq v\}$ is finite. See Apt [Apt97] and Doets [Doe94] for lucid introductions to the subject of substitutions in the context of logic programming. We will use $rng(\sigma)$ for $\{\sigma(v) \in T \mid \sigma(v) \neq v\}$, and $var(rng(\sigma))$ for $\cup\{var(\sigma(v)) \mid v \in dom(\sigma)\}$, where $var(t)$ is the set of variables of $t$.

An explicit form (or: a representation) for substitution $\sigma$ is a sequence

$$[\sigma(v_1)/v_1, \ldots, \sigma(v_n)/v_n],$$

where $\{v_1, \ldots, v_n\} = dom(\sigma)$, (i.e., $\sigma(v_i) \neq v_i$, for only the *changes* are listed), and $i \neq j$ implies $v_i \neq v_j$ (i.e., each variable in the domain is mentioned only once).

If $t \in T$ and $\sigma$ is a substitution, then $\hat{\sigma}(t)$ is given by:

$$\hat{\sigma}(v) := \sigma(v), \quad \hat{\sigma}(ft_1 \cdots t_n) := f\hat{\sigma}(t_1) \cdots \hat{\sigma}(t_n).$$

This operation in fact lifts $\sigma$ to a function in $T \to T$, but for simplicity of notation we will blur the distinction between $\sigma$ and $\hat{\sigma}$, and continue to write $\sigma$ for both.

If $\bar{t}$ is a sequence of terms, $\sigma(\bar{t})$ is the sequence that results from applying $\sigma$ to the $t_i$. Thus, we use $P\sigma\bar{t}$ for the result of applying $\sigma$ to $\bar{t}$ in $P\bar{t}$.

We will use [] for the substitution that changes nothing, i.e, [] is the only substitution $\sigma$ with $dom(\sigma) = \emptyset$. We use $\sigma, \rho, \theta$ as meta-variables ranging over substitutions.

**Definition 1 (Substitution Representations)**

$$\sigma \quad ::= \quad [] \mid [t_1/v_1, \ldots, t_n/v_n] \qquad \textit{provided } t_i \neq v_i, \textit{ and } v_i = v_j \textit{ implies } i = j.$$

We will write $\sigma^{-v_i}$ for

$$[t_1/v_1, \ldots, t_{i-1}/v_{i-1}, t_{i+1}/v_{i+1}, \ldots, t_n/v_n],$$

i.e., the result of removing the binding $\sigma v_i/v_i$ from $\sigma$. The composition $\sigma \cdot \theta$ of two substitutions $\sigma$ and $\theta$ has its usual meaning of '$\sigma$ after $\theta$', which we get by means of $\sigma \cdot \theta(v) := \sigma(\theta(v))$.

We will use $\circ$ for the *syntactic operation* of calculating the representation of $\sigma \cdot \theta$ from the representations of $\sigma$ and $\theta$. Formally:

**Definition 2 (Composition of Substitution Representations)** *Let $\sigma = [t_1/v_1, \ldots, t_n/v_n]$ and $\theta = [r_1/w_1, \ldots, r_m/w_m]$ be substitution representations. Then $\sigma \circ \theta$ is the result of removing from the sequence*

$$[\sigma(r_1)/w_1, \ldots, \sigma(r_m)/w_m, t_1/v_1, \ldots, t_n/v_n]$$

*the bindings $\sigma(r_i)/w_i$ for which $\sigma(r_i) = w_i$, and the bindings $t_j/v_j$ for which $v_j \in \{w_1, \ldots, w_m\}$.*

For example, $[x/y] \circ [y/z] = [x/z, x/y]$, $[x/z, y/x] \circ [z/x] = [x/z]$.

The following lemma, stating that the definition has the desired effect, is proved by induction on term structure:

**Lemma 3** *For all $t \in T$, for all substitution representations $\sigma$, $\theta$: $\sigma \circ \theta(t) = \sigma(\theta(t)) = \sigma \cdot \theta(t)$.*

We are now in a position to define the language $\mathcal{L}$ of DFOL, given a signature of predicate and function symbols with their arities. We distinguish between DFOL units and DFOL formulas (or sequences).

**Definition 4** *The language $\mathcal{L}$ of DFOL (given a signature of predicate and function symbols):*

$$\begin{array}{rcl} t & ::= & v \mid f\bar{t} \\ U & ::= & \sigma \mid \exists v \mid P\bar{t} \mid t_1 \doteq t_2 \mid \{\phi\} \mid \neg\{\phi\} \mid (\phi_1 \cup \phi_2) \\ \phi & ::= & U \mid U; \phi \end{array}$$

We will call formulas of the form $\{\phi\}$ *block* formulas. We will omit braces where it doesn't hurt and allow the usual abbreviations: we write $\bot$ for $\neg\{[]\}$, $\neg P\bar{t}$ for $\neg\{P\bar{t}\}$, $t_1 \neq t_2$ for $\neg\{t_1 \doteq t_2\}$, $\phi_1 \cup \phi_2$ for $(\phi_1 \cup \phi_2)$. Similarly, $\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\phi; \neg\{\psi\}\}$, $\forall v\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\exists v; \phi; \neg\{\psi\}\}$.

We can think of formula $\phi$ as built up from units $U$ by concatenation. For formula induction arguments, it is convenient to read a unit $U$ as the formula $U; []$ (recall that $[]$ is the empty substitution), thus using $[]$ for the empty list formula. In other words, we will silently add the $[]$ at the end of a formula list when we need its presence in recursive definitions or induction arguments on formula structure.

Given a first order model $\mathcal{M} = (D, I)$ for $\mathcal{L}$, the semantics of DFOL given as a binary relation on the set ${}^{V}D$, the set of all variable maps (valuations) in the domain of the model. We impose the usual non-empty domain constraint of FOL: any DFOL model $\mathcal{M} = (D, I)$ has $D \neq \emptyset$. If $s, u \in {}^{V}D$, we use $s \sim_v u$ to indicate that $s, u$ differ at most in their value for $v$, and $s \sim_X u$ to indicate that $s, u$ differ at most in their values for the members of $X$. If $s \in {}^{V}D$ and $v, v' \in V$, we use $s[v'/v]$ for the valuation $u$ given by $u(v) = s(v')$, and $u(w) = s(w)$ for all $w \in V$ with $w \neq v$.

$\mathcal{M} \models_s P\bar{t}$ indicates that $s$ satisfies the predicate $P\bar{t}$ in $\mathcal{M}$ according to the standard truth definition for classical first order logic. $[\![t]\!]^{\mathcal{M}}_s$ gives the denotation of $t$ in $\mathcal{M}$ under $s$.

If $\sigma$ is a substitution and $s$ a valuation (a member of ${}^{V}D$), we will use $s_\sigma$ for the valuation $u$ given by $u(v) = [\![\sigma(v)]\!]^{M}_s$.

**Definition 5 (Semantics of DFOL)**

$$
\begin{array}{rll}
{}_s[\![\sigma]\!]^{\mathcal{M}}_u & \textit{iff} & u = s_\sigma \\
{}_s[\![\exists v]\!]^{\mathcal{M}}_u & \textit{iff} & s \sim_v u \\
{}_s[\![P\bar{t}]\!]^{\mathcal{M}}_u & \textit{iff} & s = u \textit{ and } \mathcal{M} \models_s P\bar{t} \\
{}_s[\![t_1 \doteq t_2]\!]^{\mathcal{M}}_u & \textit{iff} & s = u \textit{ and } [\![t_1]\!]^{\mathcal{M}}_s = [\![t_2]\!]^{\mathcal{M}}_s \\
{}_s[\![\{\phi\}]\!]^{\mathcal{M}}_u & \textit{iff} & s = u \textit{ and there is a } t \textit{ with } {}_s[\![\phi]\!]^{\mathcal{M}}_t \\
{}_s[\![\neg\{\phi\}]\!]^{\mathcal{M}}_u & \textit{iff} & s = u \textit{ and there is no } t \textit{ with } {}_s[\![\phi]\!]^{\mathcal{M}}_t \\
{}_s[\![\phi_1 \cup \phi_2]\!]^{\mathcal{M}}_u & \textit{iff} & \textit{there is a } t \textit{ with } {}_s[\![\phi_1]\!]^{\mathcal{M}}_t \textit{ or } {}_s[\![\phi_2]\!]^{\mathcal{M}}_t \\
{}_s[\![U; \phi]\!]^{\mathcal{M}}_u & \textit{iff} & \textit{there is a } t \textit{ with } {}_s[\![U]\!]^{\mathcal{M}}_t \textit{ and } {}_t[\![\phi]\!]^{\mathcal{M}}_u
\end{array}
$$

Note that $\{\phi\}$ has the same relational interpretation as $\neg\neg\{\phi\}$.

The key relation we want to get to grips with in this paper is the dynamic entailment relation that is due to [GS91]:

**Definition 6 (Entailment in DFOL)** *$\phi$ dynamically entails $\psi$, notation $\phi \models \psi$, $:\Leftrightarrow$*

> *for all $\mathcal{L}$ models $\mathcal{M}$, all valuations $s, u$ for $\mathcal{M}$, if ${}_s[\![\phi]\!]^{\mathcal{M}}_u$ then there is a variable state $u'$ for which ${}_u[\![\psi]\!]^{\mathcal{M}}_{u'}$.*

## 3. Substitution in DFOL

In our definition of $\mathcal{L}$ we incorporated representations for substitutions as atoms. What we still need is a definition of the syntactic *operation* of performing a substitution on an $\mathcal{L}$ formula.

In defining the substitution operation for DFOL we have to take dynamic binding into account. In the presence of the choice operation $\cup$, there is an extra complication. What is the result of substituting $a$ for $x$ in $Px; (Qx \cup \exists x; \neg Px); Sx$? Should the $x$ in $Sx$ be replaced or not? If we follow the thread $Px, Qx, Sx$ through the formula, then it looks like the result of substitution for this thread should be $Pa, Qa, Sa$. But if we follow the thread $Px, \exists x, \neg Px, Sx$, then by the looks of it, the result of substitution for this thread should be $Pa, \exists x, \neg Px, Sx$, because the occurrence of $\exists x$ blocks further substitutions for $x$ along the thread.

The solution is to split the thread, and define substitution in such a way that

$$![a/x]Px; (Qx \cup \exists x; \neg Px); Sx$$

yields:

$$Pa; (Qa; Sa; [a/x] \cup \exists x; \neg Px; Sx).$$

Here $!\sigma\phi$ is used for the syntactic operation of *performing* $\sigma$ on $\phi$.

We can regard the substitutions as computational data, to be collected at the end of following a thread through a formula. We see that threading $[a/x]$ through the example formula yields two different substitution results at the end points: $[a/x]$ indicates that substitution $[a/x]$ is still 'active' at the end of one of the trails.

The moral of the example is that substitution in a dynamic first order language with $\cup$ must be defined relative to threads or paths, and that a reasoning system for DFOL should be set up in such a way that substitutions can be collected at the ends of computation paths. We will set up a tableau calculus for DFOL with explicit representations for substitutions at the ends of computation paths.

We define the operation $!\sigma\phi$, the result of applying substitution $\sigma$ to $\phi$, as follows:

**Definition 7 (Left-to-Right Snowball Substitution for DFOL)**

$$\begin{array}{rcl}
!\sigma\rho & := & \sigma \circ \rho \\
!\sigma(\rho;\phi) & := & !(\sigma \circ \rho)\phi \\
!\sigma(\exists v;\phi) & := & \begin{cases} \exists v; !(\sigma^{-v})\phi & \text{if } v \in dom(\sigma), v \notin var(rng(\sigma)) \\ \exists v; !\sigma\phi & \text{if } v \notin dom(\sigma), v \notin var(rng(\sigma)) \end{cases} \\
!\sigma(P\bar{t};\phi) & := & P\sigma\bar{t}; !\sigma\phi \\
!\sigma(t_1 \doteq t_2;\phi) & := & \sigma t_1 \doteq \sigma t_2; !\sigma\phi \\
!\sigma((\phi_1 \cup \phi_2);\phi_3) & := & (!\sigma(\phi_1;\phi_3) \cup !\sigma(\phi_2;\phi_3)) \\
!\sigma(\{\phi_1\};\phi_2) & := & \{!\sigma\phi_1\}; !\sigma\phi_2 \\
!\sigma(\neg\{\phi_1\};\phi_2) & := & \neg\{!\sigma\phi_1\}; !\sigma\phi_2
\end{array}$$

Note how the operation of pulling a substitution $\sigma$ through a quantifier $\exists v$ is subject to the condition that $v \notin var(rng(\sigma))$. This condition will be taken care of in the calculus for DFOL by a renaming of the quantified variable.

The reason for calling this substitution operation *snowballing* is that any substitution representations encountered on the way get carried along in the snow-slide. Note that the recursion stops at the clause for $!\sigma\rho$, for the base case $!\sigma[]$ is an instance of this.

As an example, we perform the recursive steps of the syntactic substitution mentioned at the beginning of this section:

$$\begin{array}{rl}
 & ![a/x]Px; (Qx \cup \exists x; \neg Px); Sx \\
\rightsquigarrow & Pa; ![a/x](Qx \cup \exists x; \neg Px); Sx \\
\rightsquigarrow & Pa; (![a/x](Qx; Sx) \cup ![a/x](\exists x; \neg Px; Sx)) \\
\rightsquigarrow & Pa; (Qa; Sa; ![a/x][] \cup \exists x; \neg Px; Sx; ![][]) \\
\rightsquigarrow & Pa; (Qa; Sa; [a/x] \cup \exists x; \neg Px; Sx)
\end{array}$$

As a second example, here is in essence how the tableau rules below deal with $\exists x; Px; \exists x; \neg Px; \phi$:

$$\begin{array}{rcl}
\exists x; Px; \exists x; \neg Px; \phi & \rightsquigarrow & [x_1/x]; Px; \exists x; \neg Px; \phi \\
 & = & Px_1; \exists x; \neg Px; \phi \\
 & \rightsquigarrow & Px_1 \text{ in store and process } \exists x; \neg Px; \phi \\
 & \rightsquigarrow & Px_1 \text{ in store and process } [x_2/x]; \neg Px; \phi \\
 & = & Px_1 \text{ in store and process } \neg Px_2; [x_2/x]; \phi \\
 & \rightsquigarrow & Px_1 \text{ in store, } \neg Px_2 \text{ in store and process } [x_2/x]; \phi \ldots
\end{array}$$

This syntactic substitution definition for DFOL fleshes out what has been called the 'folklore idea in dynamic logic' (Van Benthem [Ben96]) that syntactic substitution $[t/v]$ works

semantically as the program instruction $v := t$ (Goldblatt [Gol87]), with semantics given by ${}_s[\![v := t]\!]^{\mathcal{M}}_u$ iff $u = s([\![t]\!]^M_s/v)$. To see the connection, note that $v := t$ can be viewed as DFOL shorthand for $\exists v; v = t$, on the assumption that $v \notin var(t)$. To generalize this to the case where $v \in var(t)$ and to simultaneous substitution, auxiliary variables must be used. The fact that we have simultaneous substitution represented in the language saves us some bother about these.

The connection between syntactic substitution and semantic assignment is formally spelled out in the following:

**Lemma 8 (Left-to-Right Snowball Substitution Lemma for DFOL)** *For all $\mathcal{L}$ models $\mathcal{M}$, all $\mathcal{M}$-valuations $s, u$, all $\mathcal{L}$ formulas $\phi$, all substitutions $\sigma$:*

$${}_s[\![!\sigma\phi]\!]^{\mathcal{M}}_u \text{ iff } {}_s[\![\sigma; \phi]\!]^{\mathcal{M}}_u.$$

**Proof.** Induction on the structure of $\phi$. □

Immediately from this we get the following:

**Proposition 9** *DFOL has greater expressive power than DFOL with quantification replaced by definite assignment $v := d$.*

**Proof.** By the substitution lemma for DFOL, every DFOL formula with definite assignments but without quantifiers is equivalent to an $\mathcal{L}$ formula without quantifiers but with trailing substitutions. It is not difficult to see that both satisfiability and validity of quantifier free $\mathcal{L}$ formulas with substitution trails is decidable. □

In fact, the tableau system below constitutes a decision algorithm for satisfiability or validity of quantifier free $\mathcal{L}$ formulas, while the trailing substitutions summarize the finite changes made to input valuations.

Because of the switch to a dynamic setting, the usual distinction between bound and free variable occurrences has to be refined. Classically, the free variable occurrences receive their interpretation from the variable assignment, while the bound occurrences do not.

In DFOL, the single variable assignment of classical FOL gets replaced by a *pair* consisting of an input and an output assignment, so we have three classes of variable occurrences:

1. The variable occurrences that constrain the input assignment. Call these *input* occurrences.

2. The variable occurrences that constrain the output assignment. Call these *output* occurrences.

3. The variable occurrences that neither constrain the input assignment nor the output assignment. Call occurrences of this kind *classically bound*, because of their similarity to the bound variables of classical logic.

For an analysis of these and similar notions we refer to Visser [Vis98]. Here we confine ourselves to the definition of $input(\phi)$, the set of variables that have an input constraining occurrence in $\phi$ (with $\phi \in \mathcal{L}$), and $output(\phi)$, the set of variables that have an output constraining occurrence in $\phi$ (again, with $\phi \in \mathcal{L}$). Let $var(\bar{t})$ be the variables among $\bar{t}$.

**Definition 10 (Input constrained variables of $\mathcal{L}$ formulas)**

$$\begin{aligned}
input(\sigma) &:= var(rng(\sigma)) \\
input(\sigma;\phi) &:= var(rng(\sigma)) \cup (input(\phi)\backslash dom(\sigma)) \\
input(\exists v;\phi) &:= input(\phi)\backslash\{v\} \\
input(P\bar{t};\phi) &:= var(\bar{t}) \cup input(\phi) \\
input(t_1 \doteq t_2;\phi) &:= var\{t_1,t_2\} \cup input(\phi) \\
input(\{\phi_1\};\phi_2) &:= input(\phi_1) \cup input(\phi_2) \\
input(\neg\{\phi_1\};\phi_2) &:= input(\phi_1) \cup input(\phi_2) \\
input((\phi_1 \cup \phi_2);\phi_3) &:= input(\phi_1;\phi_3) \cup input(\phi_2;\phi_3).
\end{aligned}$$

The definition of the output constrained variables of $\phi$ is symmetric, but it assumes that we read formulas as stacks that grow on the right-hand sides. In programming terms, the definition demands we construct formulas with the *snoc* operator that corresponds to the *cons* operator that we used to build formulas in Definition 4. See, e.g., Bird and De Moor [BdM97]. Because the definition of formulas essentially used only concatenation, by the associativity of concatenation we have that the *snoc* counterpart to $\mathcal{L}$ looks almost exactly like $\mathcal{L}$. We will not bother to make a distinction, and we take the liberty to recurse on $\mathcal{L}$ formulas from left to right or from right to left, as the need arises. The following definition uses recursion from right to left.

**Definition 11 (Output constrained variables of $\mathcal{L}$ formulas)**

$$\begin{aligned}
output(\sigma) &:= dom(\sigma) \cup var(rng(\sigma)) \\
output(\phi;\sigma) &:= output(\phi) \cup dom(\sigma) \cup var(rng(\sigma)) \\
output(\phi;\exists v) &:= output(\phi)\backslash\{v\} \\
output(\phi;P\bar{t}) &:= var(\bar{t}) \cup output(\phi) \\
output(\phi;t_1 \doteq t_2) &:= var\{t_1,t_2\} \cup output(\phi) \\
output(\phi_2;\{\phi_1\}) &:= output(\phi_1) \cup output(\phi_2) \\
output(\phi_2;\neg\{\phi_1\}) &:= output(\phi_1) \cup output(\phi_2) \\
output(\phi_3;(\phi_1 \cup \phi_2)) &:= output(\phi_3;\phi_1) \cup output(\phi_3;\phi_2).
\end{aligned}$$

Now, the dynamic closure of a formula $\phi$ of $\mathcal{L}$ can be defined as $\exists input(\phi);\phi$. Dynamic closure of a formula is our way to make that formula insensitive to the input assignment. In a similar way, a formula $\phi$ can be made insensitive to the output assignment by postfixing $\exists output(\phi)$ to it.

The following proposition (the DFOL counterpart to the finiteness lemma from classical FOL) can be proved by induction on formula structure:

**Proposition 12** *For all $\mathcal{L}$ models $\mathcal{M}$, all valuations $s, s', u, u'$ for $\mathcal{M}$, all $\mathcal{L}$ formulas $\phi$:*

*${}_s[\![\phi]\!]^{\mathcal{M}}_u$ and $s \sim_{V\backslash input(\phi)} s'$ imply $\exists u'$ with ${}_{s'}[\![\phi]\!]^{\mathcal{M}}_{u'}$.*

*${}_s[\![\phi]\!]^{\mathcal{M}}_u$ and $u \sim_{V\backslash output(\phi)} u'$ imply $\exists s'$ with ${}_{s'}[\![\phi]\!]^{\mathcal{M}}_{u'}$.*

The right-to-left recursion on formulas from Definition 11 suggests a counterpart to the 'left to right' substitution operation we defined above.

Representations for 'right to left' substitutions are given by:

**Definition 13 (Right-to-Left Substitution Representations)**

$$\sigma^{\check{}} \quad ::= \quad [] \mid [v_1\backslash t_1, \ldots, v_n\backslash t_n] \qquad \textit{provided } t_i \neq v_i, \textit{ and } v_i = v_j \textit{ implies } i = j.$$

Right-to-left substitions are lifted to $T \to T$ functions in the same manner as their left-to-right counterparts. The interpretation of $\sigma^{\check{}}$ is given by:

$${}_s[\![\sigma^{\check{}}]\!]^{\mathcal{M}}_u \text{ iff } s = u_{\sigma^{\check{}}},$$

where $u_{\sigma^{\check{}}}$ is given by: $u_{\sigma^{\check{}}}(v) = [\![\sigma^{\check{}}(v)]\!]^{\mathcal{M}}_u$. Note that the right-to-left substitution $[v\backslash t]$ would correspond to the DFOL formula $v \doteq t; \exists v$ (on the assumption that $v \notin var(t)$).

Composition of right-to-left substitutions is the exact mirror image of $\circ$. Let $\mathcal{L}^{\check{}}$ be the result of replacing representations for left-to-right substitutions by representations for right-to-left substitutions. The definition of right-to-left snowball substitution in $\mathcal{L}^{\check{}}$ formulas, notation $\phi\sigma^{\check{}}!$, is the exact mirror image of Definition 7 above.

With right-to-left induction on $\mathcal{L}$ structure, we can now prove a right-to-left counterpart to Lemma (8):

**Lemma 14 (Right-to-Left Snowball Substitution Lemma for DFOL)** *For all $\mathcal{L}$ models $\mathcal{M}$, all valuations $s, u$ for $\mathcal{M}$, all $\mathcal{L}^{\check{}}$ formulas $\phi$, all right-to-left substitutions $\sigma^{\check{}}$:*

$${}_s[\![\phi\sigma^{\check{}}!]\!]^{\mathcal{M}}_u \textit{ iff } {}_s[\![\phi; \sigma^{\check{}}]\!]^{\mathcal{M}}_u.$$

For more on this topic, see Visser [Vis98] and [Vis00], but note that Visser's notion of substitution follows a different intuition, namely that substitution in the empty formula yields the empty formula. We think our notion is more truly dynamic, as is witnessed by the fact that it allows us to prove left-to-right and right-to-left substitution lemmas in the presence of $\cup$, which Visser's notion does not.

## 4. Modeling Computation Steps in DFOL

The illusion of symmetry in DFOL is dispelled as soon as we are prepared to face issues of knowledge and ignorance. When about to engage in a computation, we *know* the current situation (the input state), but we are *ignorant* about at least some aspect of the situations that might result from the computation (the output states). *Sinon, pas la peine de calculer.* Thus, we position ourselves firmly at the input side, and view each computation step as a step towards more knowledge of what is the case at the output side, i.e., a step towards an answer to a computation query.

A prototypical form of calculation is substitution. Carrying out substitutions is costly, so it makes sense to store substitutions for possible use later on. This is the key idea of lambda calculus (Barendregt [Bar84], or Hankin [Han94] for a concise account), for a lambda redex $(\lambda x.t)u$ suspends a substitution $[u/x]$, and the actual substitution step is made when beta reduction is performed on the redex.

In the context of dynamic predicate logic, a convenient way to suspend substitutions is by wrapping them inside formulas. This is what we have done in our definition of DFOL: in fact, we have defined a language $\mathcal{L}$ of 'DFOL formulas with explicit substitutions'.

Now, in analogy to reduction relations in lambda calculus, we define notions of reduction in DFOL. We distinguish five notions of reduction, each of them identifying a different kind of computation step that can be performed in DFOL, by defining five $R_i$ relations that describe elementary computation steps of five different kinds:

- $R_0$ describes the combination of two substitutions into a single one by means of sequential composition.
- $R_1$ describes instantiation of a predicate or equality.
- $R_2$ describes filtering of a substitution through a quantifier.
- $R_3$ describes isolating a substitution within a negation or a test.
- $R_4$ describes trail splitting at a choice point, with distribution of the substitution over the two directions.

Here are the definitions:

**Definition 15 (Notions of Reduction for $\mathcal{L}$)**

$$\begin{array}{lcl}
R_0 & := & \{(\sigma_1;\sigma_2;\phi, \sigma_1 \circ \sigma_2;\phi)\} \\
R_1 & := & \{(\sigma;P\bar{t};\phi, P\sigma\bar{t};\sigma;\phi)\} \cup \{(\sigma;t_1 \doteq t_2;\phi, \sigma t_1 \doteq \sigma t_2;\sigma;\phi)\} \\
R_2 & := & \{(\sigma;\exists v;\phi, \exists v;\sigma^{-v};\phi) \mid v \in dom(\sigma), v \notin var(rng(\sigma))\} \\
 & \cup & \{(\sigma;\exists v;\phi, \exists v;\sigma;\phi) \mid v \notin dom(\sigma), v \notin var(rng(\sigma))\} \\
R_3 & := & \{(\sigma;\neg\{\phi_1\}\phi_2, \neg\{\sigma;\phi_1\}\sigma;\phi_2)\} \cup \{(\sigma;\{\phi_1\}\phi_2, \{\sigma;\phi_1\}\sigma;\phi_2)\} \\
R_4 & := & \{(\sigma;(\phi_1 \cup \phi_2)\phi_3, (\sigma;\phi_1\phi_3 \cup \sigma;\phi_2\phi_3)\}
\end{array}$$

The following definition states what it means to make a single computation step in context:

**Definition 16 (Single Step R Reduction)**

$$\frac{(\phi_1,\phi_2) \in R}{\phi_1 \xrightarrow{R} \phi_2} \qquad \frac{\phi_1 \xrightarrow{R} \phi_2}{\{\phi_1\} \xrightarrow{R} \{\phi_2\}} \qquad \frac{\phi_1 \xrightarrow{R} \phi_2}{\neg\{\phi_1\} \xrightarrow{R} \neg\{\phi_2\}}$$

$$\frac{\phi_1 \xrightarrow{R} \phi_2}{\phi_1 \cup \psi \xrightarrow{R} \phi_2 \cup \psi} \qquad \frac{\phi_1 \xrightarrow{R} \phi_2}{\psi \cup \phi_1 \xrightarrow{R} \psi \cup \phi_2} \qquad \frac{\phi_1 \xrightarrow{R} \phi_2}{\psi;\phi_1;\chi \xrightarrow{R} \psi;\phi_2;\chi}$$

Next, we define $\xrightarrow{R}\!\!\!\!\!\rightarrow$ as the reflexive transitive closure of $\xrightarrow{R}$:

**Definition 17 (R Reduction)**

$$\frac{\phi_1 \xrightarrow{R} \phi_2}{\phi_1 \xrightarrow{R}\!\!\!\!\!\rightarrow \phi_2} \qquad \frac{}{\phi \xrightarrow{R}\!\!\!\!\!\rightarrow \phi} \qquad \frac{\phi_1 \xrightarrow{R}\!\!\!\!\!\rightarrow \phi_2 \quad \phi_2 \xrightarrow{R}\!\!\!\!\!\rightarrow \phi_3}{\phi_1 \xrightarrow{R}\!\!\!\!\!\rightarrow \phi_3}$$

Finally, $\overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow}$ gives the equivalence relation on $\mathcal{L}$ generated by taking the symmetric transitive closure of $\overset{R}{\longrightarrow\!\!\!\!\!\rightarrow}$.

**Definition 18 (R Convertibility)**

$$\frac{\phi_1 \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} \phi_2}{\phi_1 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_2} \qquad \frac{\phi_2 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_1}{\phi_1 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_2} \qquad \frac{\phi_1 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_2 \quad \phi_2 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_3}{\phi_1 \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \phi_3}$$

Let us now focus on a particular reduction relation $R$, and assume $R := R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4$.

**Lemma 19 (Computation Lemma for DFOL)**

*Suppose* $\phi \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \psi$.
*Then for all models* $\mathcal{M}$, *all valuations* $s, u$ *for* $\mathcal{M}$, ${}_s[\![\phi]\!]^{\mathcal{M}}_u$ *iff* ${}_s[\![\psi]\!]^{\mathcal{M}}_u$.

**Proof.** Induction on the length of the computation path that connects $\phi_1$ to $\phi_2$. □

The computation lemma summarizes the soundness of nearly all the computation steps that we will perform in the course of the tableau construction procedure below. We will make heavy use of it in the soundness checks for the individual tableau rules.

The reader familiar with term rewriting and abstract reduction techniques will realize that in fact we are developing the subject of $\mathcal{L}$ formula rewriting along a well trodden road. See, e.g., Baader and Nipkow [BN98]. The following results will not come as a surprise.

**Lemma 20** $\overset{R}{\longrightarrow}$ *is terminating.*

**Proof.** Define an appropriate notion of 'suspension depth' of a formula, and show that a reduction step $R_i$ decreases suspension depth. The notion of suspension depth that works is:

$$\begin{aligned}
\mathrm{sd}(\sigma) &:= 0 \\
\mathrm{sd}(\exists v) &:= 0 \\
\mathrm{sd}(P\bar{t}) &:= 0 \\
\mathrm{sd}(t_1 \doteq t_2) &:= 0 \\
\mathrm{sd}(\{\phi\}) &:= \mathrm{sd}(\phi) \\
\mathrm{sd}(\neg\{\phi\}) &:= \mathrm{sd}(\phi) \\
\mathrm{sd}(\phi_1 \cup \phi_2) &:= \max(\mathrm{sd}(\phi_1), \mathrm{sd}(\phi_2)) \\
\mathrm{sd}(\sigma; \phi) &:= \mathrm{sd}(\phi) + \mathrm{length}(\phi) \\
\mathrm{sd}((\phi_1 \cup \phi_2); \psi) &:= \max(\mathrm{sd}(\phi_1; \psi), \mathrm{sd}(\phi_2; \psi)) + 1 \\
\mathrm{sd}(U; \phi) &:= \mathrm{sd}(U) + \mathrm{sd}(\phi) \quad \text{if } U \neq \phi_1 \cup \phi_2 \text{ and } U \neq \sigma;
\end{aligned}$$

□

**Lemma 21** $\overset{R}{\longrightarrow}$ *is locally confluent.*

**Proof.** For local confluence, what we have to show is that $\phi_1 \overset{R}{\longrightarrow} \phi_2$ and $\phi_1 \overset{R}{\longrightarrow} \phi_3$ imply there is a $\psi$ with $\phi_2 \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} \psi$ and $\phi_3 \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} \psi$. This property can be verified by inspection of individual cases. □

**Theorem 22** $\xrightarrow{R}$ *is confluent.*

**Proof.** From Lemma 20 and Lemma 21 by Newman's lemma, which states that any terminating locally confluent relation is confluent. (See, e.g., Baader and Nipkov [BN98], Chapter 2.) □

From Theorem 22 it follows that $\mathcal{L}$ formulas $\phi$ have unique normal forms $\downarrow \phi$ under $\xrightarrow{R}$ reduction. It is not difficult to see that normal forms for $\mathcal{L}$ satisfy the following syntax:

$$\begin{array}{rcl} U & ::= & \exists v \mid P\bar{t} \mid t_1 \doteq t_2 \mid \{\phi\} \mid \neg\{\phi\} \\ C & ::= & \phi_1 \cup \phi_2 \\ \phi & ::= & \sigma \mid U \mid C \mid U; C \mid U; \phi \end{array}$$

Normal forms are formulas where every choice creates a split in two trails, and where substitutions occur only at the trail ends.

In fact, reduction to normal form can be achieved by means of the operation $![] : \mathcal{L} \to \mathcal{L}$ (performing a snowball substitution operation starting with the empty substitution):

**Theorem 23 (Strong Substitution Theorem for DFOL)** $![]\phi = \downarrow \phi$.

**Proof.** Check by induction that $![]\phi$ is in normal form. □

Note that $![]$ represents brute force calculation: this operation performs all the calculations suspended in a formula all the way through, in left-to-right order over the sequences, and in parallel order over the choices.

## 5. Adaptation of Tableau Reasoning to a Dynamic Setting

In classical tableau theorem proving, when investigating whether $\phi$ logically implies $\psi$, one systematically explores possibilities to make $\phi$ true and $\psi$ false. If all such explorations fail, we conclude that $\psi$ does indeed follow from $\phi$, if at least one exploration succeeds we have the makings of a counterexample, which can be read off from an open branch of a tableau in several ways (e.g., by making every fact on the true side of the tableau branch true in the model, and all other facts false, or by by making every fact on the false side of the tableau false in the model, and all other facts true; see [Ben86]).

In the course of dealing with the original $\phi$ and $\psi$ we decompose them into parts, so in general the data structure we deal with in classical tableau proving has the form $\Phi \bullet \Psi$, where $\Phi, \Psi$ are finite sets of formulas, with $\Phi$ the formulas we are committed to making true and $\Psi$ the formulas we are committed to making false. Instead of distinguishing between sets of true formulas $\Phi$ and sets of false formulas $\Psi$, we will use one-sided tableaux, with the rule for every operator $o$ matched by a $\neg o$ rule.

The tableau rule for disjunction in classical logic illustrates this. For convenience of presentation, we will assume that tableau trees grow upward, like apple trees. Thus, a tableau splitting rule like $\vee$ has the node with the disjunction $\alpha \vee \beta$ below the two branches with the disjuncts $\alpha$ and $\beta$. The rule $\vee$ serves as the 'lefthand side rule', and is matched by a rule $\neg\vee$ for dealing with the 'righthand side'. Note that we follow the customary practice of writing $\Phi + \phi$ for the set of formulas $\Phi \cup \{\phi\}$.

$$\frac{\Phi + \alpha \quad \Phi + \beta}{\Phi + \alpha \vee \beta}\ \vee \qquad\qquad \frac{\Phi + \neg\alpha + \neg\beta}{\Phi + \neg(\alpha \vee \beta)}\ \neg\vee$$

In the dynamic version of FOL, order matters: the sequencing operator ; is not commutative in general. Suppose $\Phi$ were to consist of $\exists x; Px$ and $\neg Px$. Then if we read $\Phi$ as $\exists x; Px; \neg Px$, we should get a contradiction, but if we read $\Phi$ as $\neg Px; \exists x; Px$ then the formula has a model that contains both $P$s and non-$P$s.

Suppose $\Phi$ were to consist of just $\exists x; Px; \neg\{Qx \cup Sx\}$. Then we can apply the $\neg\cup$ analogue of $\neg\vee$ to $\Phi$, but we should make sure that the results of this application, $\neg Qx$ and $\neg Sx$, remain in the scope of $\exists x; Px$. In other words, the result should be: $\exists x; Px; \neg Qx; \neg Sx$, with both $\neg Qx$ and $\neg Sx$ in the dynamic scope of the quantifier $\exists x$. In the tableau calculus to be presented, we will ensure that negation rules $\neg o$ take dynamic context into account, and that all formulas come with an appropriate binding context, to be supplied by explicit substitutions.

*Storage of Computed Substitutions* Since performing substitutions may be expensive, it will be advantageous to only perform a substitution $\sigma$ on $\phi$ when needed. Rather than compute $!\sigma\phi$, the tableau rules will store $\sigma; \phi$, and compute the substitution in single steps as the need arises.

The tableau method is a *confutation* method. Tableau theorem proving can be viewed as a process of gradually building a domain $D$ and working out requirements to be imposed on that domain. The tableau procedure that investigates whether $\phi$ dynamically implies $\psi$ will build a domain with positive and negative facts.

We will use domain names from $V$, plus names $v_n$, with $v \in V$, $n \in \mathbb{N}^+$ (i.e., we implicitly equate $v$ with $v_0$). In an implementation one would in general keep track of the current domain by means of a function $N : V \to \mathbb{N}$, but in our presentation below we will use $v_{\text{current}}$ for the current instance of the dynamic variable $v$, and $v_{\text{new}}$ as a generic renaming of the variable $v$. What this boils down to is that we build the terms of the tableau language over a variable set $V \times \mathbb{N}$, using $v_0 (= \langle v, 0\rangle), v_1, v_2, \ldots$ for the various dynamic instances of $v$.

The current domain always consists of all the terms occurring in the current database of literals (predicate and equality atoms and their negations). We will use $t \in \Phi$ for: '$t$ is a term occurring in a literal in $\Phi$'.

*Structure of Tableau Nodes* A DFOL tableau node is a set $\Phi$ of formulas. $\Phi$ will be of the form:

$$C + \phi + \{\phi_1\} + \cdots + \{\phi_n\} + \neg\{\phi_{n+1}\} + \cdots + \neg\{\phi_{n+m}\},$$

with $C$ a set of literals (atomic formulas or their negations), and $n, m \geq 0$. Thus, $\phi$ is the only member of $\Phi$ that can have an external dynamic effect, for the literals and the $\{\phi_i\}$ and $\neg\{\phi_j\}$ are test formulas.

Without loss of generality, we can assume that $\phi$ is of the form $\sigma; \phi'$, and similarly for the $\{\phi_i\}$ and $\neg\{\phi_j\}$, by prefixing with the empty substitution if the need arises.

Any tableau node can be thought of as a database $\Phi$ of formulas true at that node. Because our databases may contain identities, we need some preliminary definitions in order to define closure of a database.

Note that the variables occurring in literals in the database are not universally quantified: occurrence of $Pv_2$ in a database does *not* mean that everything has property $P$, but rather that at least one thing has $P$. Thus, the variables occurring in literals can be taken as *names*,

and the terms occurring in literals can be considered as *ground terms*. This means that we can determine closure of a database in terms of the *congruence closure* of the set of equalities occurring in a database $\Phi$. See [BN98], Chapter 4, also for what follows.

**Definition 24** *The congruence closure of $\Phi$, notation $\approx_\Phi$, is the smallest congruence on $T$ that contains all the equalities in $\Phi$.*

In general, $\approx_\Phi$ will be infinite: if $a \doteq b$ is an equality in $\Phi$, and $f$ is a one place function symbol in the language, then $\approx_\Phi$ will contain $fa \doteq fb, ffa \doteq ffb, fffa \doteq fffb, \ldots$. Therefore, we must use congruence closure modulo some finite set instead.

**Definition 25** *Let $S$ be the set of all subterms of terms occurring in a literal in $\Phi$. (Recall: subterms need not be proper.) Then the congruence closure of $\Phi$ modulo $S$, notation $CC_S(\Phi)$, is the finite set of equalities $\approx_\Phi \cap (S \times S)$.*

We can decide whether $t \doteq t'$ in $\mathrm{CC}_S(\Phi)$; Baader and Nipkow give an algorithm for computing $\mathrm{CC}_S(G)$, for finite sets of equalities $G$ and terms $S$, in polynomial time.

**Definition 26** *$t \approx t'$ is suspended in $\Phi$ if $t \doteq t' \in CC_S(\Phi)$, where $S$ is the set of all subterms of terms occurring in literals in $\Phi$. We extend this notation to sequences: $\bar{t} \approx \bar{t}'$ is suspended in $\Phi$ if $t_1 \approx t'_1, \ldots, t_n \approx t'_n$ are suspended in $\Phi$.*

**Definition 27** *A formula set $\Phi$ is closed if either $\neg\{\sigma\} \in \Phi$ (recall that $\bot$ is an abbreviation for $\neg\{[]\}$), or for some $\bar{t} \approx \bar{t}'$ suspended in $\Phi$ we have $P\bar{t} \in \Phi$, $\neg P\bar{t}' \in \Phi$, or for a pair of terms $t_1, t_2$ with $t_1 \approx t_2$ suspended in $\Phi$ we have $t_1 \neq t_2 \in \Phi$. A tableau node is* closed *if its formula set (database) is closed. A tableau is closed if all of its leaf nodes are closed.*

The tableau rules specify a procedure for extending a tableau tree with new leaf nodes. The tableau rules specify a *procedure*, but not an *algorithm*, for tableau tree construction: as in the tableau systems for classical FOL, there is no guarantee of termination.

As in classical FOL tableau proving, it might happen that no rule can be applied because the initial domain is empty. In DFOL tableau theorem proving this happens when the only formulas to which a rule can be applied are of the form $\neg\{\exists v; \phi\}$. In such a case it is allowed to assume the presence of a pre-given element $v$ in the domain, and continue with $\neg\{\phi\}$. This takes care of the non-empty domain constraint that DFOL shares with classical FOL: the assumption that any $\mathcal{L}$ model $\mathcal{M}$ has a non-empty domain.

Our inquiry whether $\phi$ dynamically implies $\psi$ starts with a formula $\phi; \neg\{\psi\}$, so when we prefix with the empty computed substitution [] the initial configuration looks like this:

$$[]; \phi; \neg\{\psi\}.$$

In the course of the tableau construction process, we let the computed substitution grow.

In the tableau rule for quantification we will need renaming of variables:

**Definition 28 (Variable Renaming in Terms and Formulas)**

$$
\begin{array}{rcl}
w^v_{v'} & := & \begin{cases} v' & \textit{if } v \equiv w, \\ w & \textit{otherwise} \end{cases} \\
(ft_1 \cdots t_n)^v_{v'} & := & ft_1{}^v_{v'} \cdots t_n{}^v_{v'} \\
[]^v_{v'} & := & [v'/v] \\
(\sigma; \phi)^v_{v'} & := & [v'/v] \circ \sigma; \phi \\
(\exists w; \phi)^v_{v'} & := & \begin{cases} \exists w; \phi & \textit{if } w \equiv v, \\ \exists w; \phi^v_{v'} & \textit{otherwise} \end{cases} \\
(Pt_1 \cdots t_n; \phi)^v_{v'} & := & Pt_1{}^v_{v'} \cdots t_n{}^v_{v'}; \phi^v_{v'} \\
(t_1 \doteq t_2; \phi)^v_{v'} & := & t_1{}^v_{v'} \doteq t_2{}^v_{v'}; \phi^v_{v'} \\
(\{\phi\}; \psi)^v_{v'} & := & \{\phi^v_{v'}\}\psi^v_{v'} \\
(\neg\{\phi\}; \psi)^v_{v'} & := & \neg\{\phi^v_{v'}\}\psi^v_{v'} \\
((\phi \cup \psi); \chi)^v_{v'} & := & (\phi; \chi)^v_{v'} \cup (\psi; \chi)^v_{v'}.
\end{array}
$$

**Lemma 29** ${}_s[\![[v'/v]; \phi]\!]^{\mathcal{M}}_u$ *iff* ${}_s[\![\phi^v_{v'}]\!]^{\mathcal{M}}_u$.

**Proof.** Induction on the structure of $\phi$, using the definition of the renaming operation. For the base case, observe that $[v'/v] \circ [] = [v'/v] = []^v_{v'}$. □

When using $\phi^v_{v'}$ we will have to ensure that $v'$ does not get dynamically bound in $\phi$. We can do this by always taking $v'$ fresh.

Lemma 29 tells us that $\phi^v_{v'}$ and $[v'/v]; \phi$ have the same semantics. A good reason for using $\phi^v_{v'}$ rather than $[v'/v]; \phi$ is that we do not wish to collect all renamings of $v$ in a computed substitution. Also, a renaming $\phi^v_{v'}$ can be useful to separate the free occurrences of $v$ in $\phi$ from the bound occurrences, in order to prevent accidental binding under substitution. Consider example (5.1).

$$y < x; \neg\{\exists x; \exists y; x < y\}. \tag{5.1}$$

This is contradictory, but to derive the contradiction one has to swap $x$ and $y$ in the subformula $x < y$ of (5.1). Using a step by step approach of composition of bindings for $x$ and $y$, this cannot be done. If, instead, we first rename the free occurrences of $x, y$ to $x_1, y_1$, there is no problem for the step by step approach, for now one just has to substitute $y_1$ for $x$ and $x_1$ for $y$, which is easy: $y_1 < x_1; \neg\{\exists x; \exists y; x < y\}$ gives $\neg\{[y_1/x]; \exists y; x < y\}$, and next $\neg\{[y_1/x] \circ [x_1/y]; x < y\}$, which gives $\neg y_1 < x_1$, and contradiction.

We need one more operation on substitution representations: replacement of terms in the range of $\sigma$. If $\sigma$ is a substitution representation, then $\sigma^t_{t'}$ be the substitution representation $[r_1/v_1, \ldots, r_n/v_n]$, where $\{v_1, \ldots, v_n\} = \mathit{dom}(\sigma)$, and each $r_i$ is the result of replacing all occurrences of $t$ in $\sigma(v_i)$ by occurrences of $t'$. For example, $[f(a)/x, b/y, a/z]^a_b = [f(b)/x, b/y, b/z]$.

## 6. Tableau Rules for DFOL

Define $\mathbf{C}(\Phi)$, for $\Phi$ a set of DFOL formulas, as follows:

**Definition 30** *Let $\Phi$ be a set of $\mathcal{L}$ formulas. Then $\mathbf{C}(\Phi)$ iff there is an $\mathcal{L}$ model $\mathcal{M}$ and a valuation $s$ for $\mathcal{M}$ such that for every $\phi \in \Phi$ there is a valuation $u$ with ${}_s[\![\phi]\!]^{\mathcal{M}}_u$.*

Figure 1: The Calculus for DFOL

| | |
|---|---|
| closure | $\dfrac{}{P\bar{t} + \neg P\bar{t}' + \Phi}\ \bar{t} \approx \bar{t}'$ susp in $\Phi$ $\qquad$ $\dfrac{}{t \neq t' + \Phi}\ t \approx t'$ susp in $\Phi$ $\qquad$ $\dfrac{}{\neg\{\sigma\} + \Phi}$ |
| composition | $\dfrac{\sigma \circ \rho; \phi + \Phi}{\sigma; \rho; \phi + \Phi}$ $\qquad$ $\dfrac{\{\sigma \circ \rho; \phi\} + \Phi}{\{\sigma; \rho; \phi\} + \Phi}$ $\qquad$ $\dfrac{\neg\{\sigma \circ \rho; \phi\} + \Phi}{\neg\{\sigma; \rho; \phi\} + \Phi}$ |
| predicate | $\dfrac{P\sigma\bar{t} + \sigma; \phi + \Phi}{\sigma; P\bar{t}; \phi + \Phi}$ $\qquad$ $\dfrac{P\sigma\bar{t} + \{\sigma; \phi\} + \Phi}{\{\sigma; P\bar{t}; \phi\} + \Phi}$ $\qquad$ $\dfrac{\neg P\sigma\bar{t} + \Phi \qquad \neg\{\sigma; \phi\} + \Phi}{\neg\{\sigma; P\bar{t}; \phi\} + \Phi}$ |
| equality (1) | $\dfrac{v \doteq \sigma t_2 + [\sigma t_2/v] \circ \sigma; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_1 \equiv v$ $\qquad$ $\dfrac{\sigma t_1 \doteq v + [\sigma t_1/v] \circ \sigma; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_2 \equiv v$ |
| equality (2) | $\dfrac{\sigma t_1 \doteq \sigma t_2 + \sigma^{\sigma t_1}_{\sigma t_2}; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_1$ not a variable, $\sigma t_2$ not a variable, |
| equality (3) | $\dfrac{v \doteq \sigma t_2 + \{[\sigma t_2/v] \circ \sigma; \phi\} + \Phi}{\{\sigma; t_1 \doteq t_2; \phi\} + \Phi}\ \sigma t_1 \equiv v$ $\qquad$ $\dfrac{\sigma t_1 \doteq v + \{[\sigma t_1/v] \circ \sigma; \phi\} + \Phi}{\{\sigma; t_1 \doteq t_2; \phi\} + \Phi}\ \sigma t_2 \equiv v$ |
| equality (4) | $\dfrac{\sigma t_1 \doteq \sigma t_2 + \sigma^{\sigma t_1}_{\sigma t_2}; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_1$ not a variable, $\sigma t_2$ not a variable, |
| equality (5) | $\dfrac{\sigma t_1 \neq \sigma t_2 + \Phi \qquad \neg\{\sigma; \phi\} + \Phi}{\neg\{\sigma; t_1 \doteq t_2; \phi\} + \Phi}$ |
| quantifier | $\dfrac{\sigma; \phi^v_{v_{\text{new}}} + \Phi}{\sigma; \exists v; \phi + \Phi}$ $\qquad$ $\dfrac{\{\sigma; \phi^v_{v_{\text{new}}}\} + \Phi}{\{\sigma; \exists v; \phi\} + \Phi}$ $\qquad$ $\dfrac{\neg\{\sigma; \exists v; \phi\} + \neg\{\sigma \circ [t/v]; \phi\} + \Phi}{\neg\{\sigma; \exists v; \phi\} + \Phi}\ t \in \Phi$ |
| choice (1) | $\dfrac{\sigma; \phi_1; \phi_3 + \Phi \qquad \sigma; \phi_2; \phi_3 + \Phi}{\sigma; (\phi_1 \cup \phi_2); \phi_3 + \Phi}$ $\qquad$ $\dfrac{\{\sigma; \phi_1; \phi_3\} + \Phi \qquad \{\sigma; \phi_2; \phi_3\} + \Phi}{\{\sigma; (\phi_1 \cup \phi_2); \phi_3\} + \Phi}$ |
| choice (2) | $\dfrac{\neg\{\sigma; \phi_1; \phi_3\} + \neg\{\sigma; \phi_2; \phi_3\} + \Phi}{\neg\{\sigma; (\phi_1 \cup \phi_2); \phi_3\} + \Phi}$ |
| negation | $\dfrac{\{\sigma; \phi_1\} + \Phi \qquad \neg\{\sigma; \phi_2\} + \Phi}{\neg\{\sigma; \neg\{\phi_1\}; \phi_2\} + \Phi}$ |
| distrib (1) | $\dfrac{\{\sigma; \phi_1\} + \sigma; \phi_2 + \Phi}{\sigma; \{\phi_1\}; \phi_2 + \Phi}$ $\qquad$ $\dfrac{\neg\{\sigma; \phi_1\} + \sigma; \phi_2 + \Phi}{\sigma; \neg\{\phi_1\}; \phi_2 + \Phi}$ |
| distrib (2) | $\dfrac{\{\sigma; \phi_1\} + \{\sigma; \phi_2\} + \Phi}{\{\sigma; \{\phi_1\}; \phi_2\} + \Phi}$ $\qquad$ $\dfrac{\neg\{\sigma; \phi_1\} + \{\sigma; \phi_2\} + \Phi}{\{\sigma; \neg\{\phi_1\}; \phi_2\} + \Phi}$ |

We will show as we go along that the tableau rules preserve the $\mathbf{C}(\Phi)$ relation, in the following sense:

- if $\Phi$ yields $\Phi'$ by a unary tableau rule application, then $\mathbf{C}(\Phi)$ implies $\mathbf{C}(\Phi')$.
- if $\Phi$ yields $\Phi'$ and $\Phi''$ by a splitting tableau rule application, then $\mathbf{C}(\Phi)$ implies both $\mathbf{C}(\Phi')$ and $\mathbf{C}(\Phi'')$.

From this soundness of the individual tableau rules follows the soundness of the tableau calculus: see Theorem 31.

Key to the calculus is the theory of explicit substitution developed in Section 4. Note that we can take the form of any DFOL formula to be $\sigma;\phi$ (prefix [] to $\phi$ when the need arises). The tableau rules have the effect that substitutions get pushed from left to right in the tableaus, and appear as computed results at the open nodes, according to the principles explained in Section 4. A summary of the calculus is given in Figure 1.

*Closure*

$$\frac{}{P\bar{t} + \neg P\bar{t}' + \Phi}\ \bar{t} \approx \bar{t}' \text{ susp in } \Phi \qquad \frac{}{t \neq t' + \Phi}\ t \approx t' \text{ susp in } \Phi \qquad \frac{}{\neg\{\sigma\} + \Phi}$$

Soundness: if $\bar{t}$ and $\bar{t}'$ can be unified by a list of equalities suspended in $\Phi$ then $P\bar{t}, \neg P\bar{t}', \Phi$ has no models. Similarly, if the equality of $t, t'$ is suspended in $\Phi$, then $t \neq t', \Phi$ has no models. Finally, since substitutions always succeed, $\neg\{\sigma\}$ is inconsistent.

*Composition*

$$\frac{\sigma \circ \rho; \phi + \Phi}{\sigma; \rho; \phi + \Phi} \qquad \frac{\{\sigma \circ \rho; \phi\} + \Phi}{\{\sigma; \rho; \phi\} + \Phi} \qquad \frac{\neg\{\sigma \circ \rho; \phi\} + \Phi}{\neg\{\sigma; \rho; \phi\} + \Phi}$$

Soundness: substitutions always succeed, and the result of first performing substitution $\rho$ and next substitution $\sigma$ is the same as that of performing the single substitution $\sigma \circ \rho$.

*Predicate*

$$\frac{P\sigma\bar{t} + \sigma; \phi + \Phi}{\sigma; P\bar{t}; \phi + \Phi} \qquad \frac{P\sigma\bar{t} + \{\sigma; \phi\} + \Phi}{\{\sigma; P\bar{t}; \phi\} + \Phi} \qquad \frac{\neg P\sigma\bar{t} + \Phi \qquad \neg\{\sigma; \phi\} + \Phi}{\neg\{\sigma; P\bar{t}; \phi\} + \Phi}$$

Predicates are detached and shifted to the database. Block structure, if present, is preserved. Predicates under negation give rise to branching. It is easily checked that these rules are sound.

*Equality*

$$\frac{v \doteq \sigma t_2 + [\sigma t_2/v] \circ \sigma; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_1 \equiv v \qquad \frac{\sigma t_1 \doteq v + [\sigma t_1/v] \circ \sigma; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi}\ \sigma t_2 \equiv v$$

Here are some example applications of these rules:

$$\frac{\dfrac{y \doteq a + [a/y, a/x]; \phi + \Phi}{y \doteq a + [a/y] \circ [a/x]; \phi + \Phi}}{[a/x]; y \doteq x; \phi + \Phi} \qquad \frac{\dfrac{fy \doteq y + [fy/y, fy/x]; \phi + \Phi}{fy \doteq y + [fy/y] \circ [y/x]; \phi + \Phi}}{[y/x]; fx \doteq y; \phi + \Phi}$$

For the case where both sides of an equality get mapped to a non-variable term, we need a different rule, for now we replace a term $\sigma t_1$ in the range of a substitution by another term $\sigma t_2$.

$$\frac{\sigma t_1 \doteq \sigma t_2 + \sigma^{\sigma t_1}_{\sigma t_2}; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi} \ \sigma t_1 \text{ not a variable, } \sigma t_2 \text{ not a variable,}$$

An example application of this:

$$\frac{\dfrac{fx \doteq fy + [fy/x, fy/y]; \phi + \Phi}{fx \doteq fy + [fx/x, fy/y]^{fx}_{fy}; \phi + \Phi}}{[fx/x, fy/y]; x \doteq y; \phi + \Phi}$$

Soundness: adding $\sigma t_1 \doteq \sigma t_2$ to the database represents storage of the meaning of $t_1 \doteq t_2$ under $\sigma$. In addition, the rules use this information to update the current substitution, if possible. This computational treatment of equality statements was inspired by [AB99].

*Equality under Block* Similar to previous equality rules:

$$\frac{v \doteq \sigma t_2 + \{[\sigma t_2/v] \circ \sigma; \phi\} + \Phi}{\{\sigma; t_1 \doteq t_2; \phi\} + \Phi} \ \sigma t_1 \equiv v \qquad \frac{\sigma t_1 \doteq v + \{[\sigma t_1/v] \circ \sigma; \phi\} + \Phi}{\{\sigma; t_1 \doteq t_2; \phi\} + \Phi} \ \sigma t_2 \equiv v$$

$$\frac{\sigma t_1 \doteq \sigma t_2 + \sigma^{\sigma t_1}_{\sigma t_2}; \phi + \Phi}{\sigma; t_1 \doteq t_2; \phi + \Phi} \ \sigma t_1 \text{ not a variable, } \sigma t_2 \text{ not a variable,}$$

Soundness: as for equality.

*Equality under Negation*

$$\frac{\sigma t_1 \neq \sigma t_2 + \Phi \qquad \neg\{\sigma; \phi\} + \Phi}{\neg\{\sigma; t_1 \doteq t_2; \phi\} + \Phi}$$

Soundness: obvious.

*Quantifier*

$$\frac{\sigma; \phi^v_{v_{\text{new}}} + \Phi}{\sigma; \exists v; \phi + \Phi} \qquad \frac{\{\sigma; \phi^v_{v_{\text{new}}}\} + \Phi}{\{\sigma; \exists v; \phi\} + \Phi} \qquad \frac{\neg\{\sigma; \exists v; \phi\} + \neg\{\sigma \circ [t/v]; \phi\} + \Phi}{\neg\{\sigma; \exists v; \phi\} + \Phi} \ t \in \Phi$$

The effect of quantifier $\exists v$ is that it cuts the dynamic link between currently dynamic incarnations of $v$ (occurrences of $v_{\text{current}}$), and occurrences to come. Occurrences of $v$ in the dynamic scope of $\exists v$ are renamed to $v_{\text{new}}$ in the trailing formula $\phi$.

**Note 1** The reason for using $\phi^v_{v_{\text{new}}}$ rather than $[v_{\text{new}}/v]; \phi$ for the renaming, is that it does not make computational sense to collect all the successive renamings of a variable in the computed substitution.

**Note 2** In an implementation, bindings for $v_{\text{current}}$, the current renaming of the dynamic variable $v$, can be removed from the computed substitution, by switching to the substitution $\sigma^{-v_{\text{current}}}$, since they will not be needed further on.

Soundness of quantifier under negation: to make $\exists v; \phi$ false at the current node, under $\sigma$, we have to make $\phi$ false for any $t$ currently in the database, and for all terms $t$ to be added later on.

*Important remark* Quantification under negation is the only rule of the calculus that does not decompose its target formula. The possibility of repeatedly triggering this rule for the same target formula, as the domain grows, is what may cause the tableau building process to loop. This is not spurious repetition, for the rule is triggered for each of the domain elements. This situation is completely analogous to the case for standard FOL.

*Choice*

$$\frac{\sigma;\phi_1;\phi_3+\Phi \qquad \sigma;\phi_2;\phi_3+\Phi}{\sigma;(\phi_1\cup\phi_2);\phi_3+\Phi} \qquad\qquad \frac{\{\sigma;\phi_1;\phi_3\}+\Phi \qquad \{\sigma;\phi_2;\phi_3\}+\Phi}{\{\sigma;(\phi_1\cup\phi_2);\phi_3\}+\Phi}$$

Soundness: immediate from the computation lemma for DFOL. Note how the rules effectively split the computation, with distribution of $\sigma$ over the two trails.

*Choice under Negation*

$$\frac{\neg\{\sigma;\phi_1;\phi_3\}+\neg\{\sigma;\phi_2;\phi_3\}+\Phi}{\neg\{\sigma;(\phi_1\cup\phi_2);\phi_3\}+\Phi}$$

Soundness: immediate from the computation lemma for DFOL.

*Negation under Negation*

$$\frac{\{\sigma;\phi_1\}+\Phi \qquad \neg\{\sigma;\phi_2\}+\Phi}{\neg\{\sigma;\neg\{\phi_1\};\phi_2\}+\Phi}$$

Soundness: immediate from the semantics of block statements, and the fact that substitutions distribute over blocks. Note that the rule for double negation follows immediately from this: see Section 7.

*Distribution of Substitution over Block and over Negation*

$$\frac{\{\sigma;\phi_1\}+\sigma;\phi_2+\Phi}{\sigma;\{\phi_1\};\phi_2+\Phi} \qquad\qquad \frac{\neg\{\sigma;\phi_1\}+\sigma;\phi_2+\Phi}{\sigma;\neg\{\phi_1\};\phi_2+\Phi}$$

Soundness: immediate from the computation lemma for DFOL.

*Distribution of Substitution over Block and Negation, Inside Block*

$$\frac{\{\sigma;\phi_1\}+\{\sigma;\phi_2\}+\Phi}{\{\sigma;\{\phi_1\};\phi_2\}+\Phi} \qquad\qquad \frac{\neg\{\sigma;\phi_1\}+\{\sigma;\phi_2\}+\Phi}{\{\sigma;\neg\{\phi_1\};\phi_2\}+\Phi}$$

Soundness: immediate from the fact that $\{\phi\}$ and $\{\{\phi\}\}$ have the same dynamic meaning, and from the fact that $\neg\{\phi\}$ and $\{\neg\{\phi\}\}$ have the same dynamic meaning.

*That's all* This completes the presentation of the tableau calculus for DFOL. We have checked the rules for soundness as we went on, so we have established the following:

**Theorem 31 (Soundness Theorem)** *The tableau calculus for DFOL is sound:*

*If the tableau for $\phi;\neg\{\psi\}$ closes then $\phi\models\psi$.*

**Proof.** If the tableau for $\phi; \neg\{\psi\}$ closes, then there are no $\mathcal{M}, s, s'$ with ${}_s[\![\phi; \neg\{\psi\}]\!]^{\mathcal{M}}_{s'}$. In other words: for every $\mathcal{L}$ model $\mathcal{M}$ and every pair of variable states $s, u$ for $\mathcal{M}$ with ${}_s[\![\phi]\!]^{\mathcal{M}}_u$ there has to be a variable state $u'$ with ${}_u[\![\psi]\!]^{\mathcal{M}}_{u'}$, i.e., we have $\phi \models \psi$, in the sense of Definition 6. □

## 7. Derived Principles

*Double Negation* The rule for double negation,

$$\frac{\{\phi\} + \Phi}{\neg\neg\{\phi\} + \Phi}$$

is derived from the rule for negation under negation and the closure rule for negated substitutions:

$$\frac{\{[]; \phi\} + \Phi \qquad \overline{\neg\{[]\} + \Phi}}{\neg\{[]; \neg\{\phi\}\} + \Phi}$$

*Blocks Detachment* A sequence of blocks $\pm\{\phi_1\}; \ldots ; \pm\{\phi_n\}$, where $\pm\{\phi_i\}$ is either $\{\phi_i\}$ or $\neg\{\phi_i\}$, yields the set of its components, by a series of applications of distribution of the empty substitution over block or negation. This is useful, as the list $\pm\{\phi_1\}, \ldots , \pm\{\phi_n\}$ can be processed in any order (because every $\pm\{\phi_i\}$ is a test).

In a schema:

$$\frac{\pm\{\phi_1\}, \ldots , \pm\{\phi_n\}}{\pm\{\phi_1\}; \ldots ; \pm\{\phi_n\}}$$

*Nonatomic Closure*

**Theorem 32** *The following closure axioms are admissible in the calculus:*

$$\overline{\phi + \neg\{\phi\} + \Phi} \qquad\qquad \overline{\{\phi\} + \neg\{\phi\} + \Phi}$$

**Proof.** Induction on the complexity of the tableau trees for $\phi, \neg\{\phi\}$ and for $\{\phi\}, \neg\{\phi\}$. □

*Negation Splitting*

**Theorem 33** *The following rules are admissible in the calculus:*

$$\frac{\{\phi; \{\psi\}\} \quad \neg\{\phi; \chi\}}{\neg\{\phi; \neg\{\psi\}; \chi\}} \qquad\qquad \frac{\{\phi; \neg\{\psi\}\} \quad \neg\{\phi; \chi\}}{\neg\{\phi; \{\psi\}; \chi\}}$$

**Proof.** Induction on the complexity of $\phi$, starting with the base case where $\phi$ equals $\sigma$, and using the fact that $\neg\{\psi\}$ and $\{\psi\}$ are tests. □

Negation splitting can be viewed as the DFOL guise of a well known principle from modal logic: $\Box(A \vee B) \to (\Diamond A \vee \Box B)$. To see the connection, note that $\neg\{\phi; \neg\{\psi\}; \chi\}$ is semantically equivalent to $\neg\{\phi; \neg(\psi \cup \neg\{\chi\})\}$, where $\neg\{\phi; \neg \cdots\}$ behaves as a $\Box$ modality.

## 8. Examples

*Syllogistic Reasoning* Consider the syllogism:

$$\forall x(Ax \to Bx), \forall x(Bx \to Cx) / \forall x(Ax \to Cx).$$

Its DFOL guise is:

$$\forall x\{Ax \Rightarrow Bx\}, \forall x\{Bx \Rightarrow Cx\} \ / \ \forall x\{Ax \Rightarrow Cx\}.$$

This is in turn an abbreviation of:

$$\neg\{\exists x; Ax; \neg Bx\}, \neg\{\exists x; Bx; \neg Cx\} \ / \ \neg\{\exists x; Ax; \neg Cx\}$$

DFOL tableau (with ellipsis for repeated information):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Ax_1, \neg Cx_1, \neg Ax_1 + \cdots} \quad \dfrac{\dfrac{}{Ax_1, \neg Cx_1, Bx_1, \neg Bx_1 + \cdots} \quad \dfrac{}{Ax_1, \neg Cx_1, Bx_1, Cx_1 + \cdots}}{Ax_1, \neg Cx_1, Bx_1, \neg\{[x_1/x]; Bx; \neg Cx\} + \cdots}}{Ax_1, \neg Cx_1, \neg\{[x_1/x]; Ax; \neg Bx\} + \cdots}}{Ax_1, \neg Cx_1 + \cdots}}{\{Ax_1; \neg Cx_1\} + \cdots}}{\{\exists x; Ax; \neg Cx\}, \neg\{\exists x; Ax; \neg Bx\}, \neg\{\exists x; Bx; \neg Cx\}}}{\neg\{\exists x; Ax; \neg Bx\}; \neg\{\exists x; Bx; \neg Cx\}; \{\exists x; Ax; \neg Cx\}}$$

*Dynamic Donkey Reasoning* The hackneyed example for dynamic binding in natural language, *If a farmer owns a donkey, he beats it*, has the following DFOL shape:

$$\{\exists x; \exists y; Fx; Dy; Oxy \Rightarrow Bxy\},$$

which is shorthand for:

$$\neg\{\exists x; \exists y; Fx; Dy; Oxy; \neg Bxy\}.$$

Here is how to draw conclusions from this in a DFOL tableau calculation. Consider the natural language text: *If a farmer owns a donkey, he beats it. Alfonso is a farmer and owns a donkey.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Fa} \quad \dfrac{\dfrac{}{Dz_1} \quad \dfrac{\dfrac{}{\neg Oaz_1} \quad \dfrac{\dfrac{Baz_1}{\{[a/x, z_1/y]; Bxy\}}}{\neg\{[a/x, z_1/y]; \neg Bxy\}}}{\neg\{[a/x, z_1/y]; Oxy; \neg Bxy\}}}{\neg\{[a/x, z_1/y]; Dy; Oxy; \neg Bxy\}}}{\neg\{[a/x, z_1/y]; Fx; Dy; Oxy; \neg Bxy\}}}{Fa, Dz_1, Oaz_1}}{\neg\{\exists x; \exists y; Fx; Dy; Oxy; \neg Bxy\}; Fa; \exists z; Dz; Oaz}$$

The open tableau branch yields the fact $Baz_1$, plus the following further information about $z_1$: $Dz_1, Oaz_1$. This further information is useful to identify $z_1$ as *the donkey that Alfonso owns* (or perhaps *a donkey that Alfonso owns*) that was introduced in the text.

*Open Tableau Branches, Partial Models, Reference Resolution* An open tableau branch for a DFOL formula $\phi$ may be viewed as a *partial model* for $\phi$, with just enough information to verify the formula. For instance, the open branch in the previous example does not specify whether donkey $z_1$ also beats Alfonso or not: $Bz_1a$ is neither among the facts (true atoms) nor among the negated facts (false atoms) of the branch.

In tableau branches involving equality there is also another kind of partiality involved: the terms are *proto-objects* rather than genuine objects, in sense that they have not yet 'made up their minds' about which individual they are: two terms $t_1, t_2$ on a tableau that does not contain $t_1 \neq t_2$ may be interpreted as a single individual. This is because the information about equality that the branch provides is also partial.

The level of tableau style generation of partial models for discourse may be just the right level for pronoun reference resolution (cf. the suggestion in [BvE82]). Since reference resolution is a processing step that links a pronoun to a suitable antecedent, what about equating the suitable antecedents with the available terms of the branches in a tableau? After all, reference resolution for pronouns is part of semantic processing, so it has a more natural habitat at the level of processing NL representations than at the level of mere represention of NL meaning.

Building on this idea, we (tentatively) introduce the following rule for pronoun resolution:

$$\frac{P(t) + \Phi}{P(\text{pro}) + \Phi}\ t \in \Phi \qquad \frac{\neg P(t) + \Phi}{\neg P(\text{pro}) + \Phi}\ t \in \Phi$$

We demonstrate the rule for the following piece of discourse.

Every farmer owns a donkey. Some farmer beats it. (8.1)

$$\dfrac{\dfrac{\dfrac{\dfrac{Fz_1, Bz_1y_1, Dy_1, Oz_1y_1}{Fz_1, Bz_1 \text{ it}, Dy_1, Oz_1y_1}}{Fz_1, Bz_1 \text{ it}, \{Dy_1; Oz_1y_1\}}}{\dfrac{\neg Fz_1}{Fz_1, Bz_1 \text{ it}, \{\exists y; Dy; Oz_1y\}}}}{\dfrac{Fz_1, Bz_1 \text{ it}, \neg\{Fz_1; \neg\{\exists y; Dy; Oz_1y\}\}}{\dfrac{Fz_1; Bz_1 \text{ it}, \neg\{\exists x; Fx; \neg\{\exists y; Dy; Oxy\}\}}{\forall x\{Fx \Rightarrow \exists y; Dy; Oxy\}; \exists z; Fz; Bz \text{ it}}}}$$

Intuitively, the following happens. First, a term $z_1$ introduced for *Some farmer*. This leads to an unresolved fact '$Bz_1$ it' in the database of the partial model under construction. Later, the pronoun *it* is resolved to 'the donkey that $z_1$ owns' generated from *every farmer owns a donkey*, and represented in the database of the partial model as $y_1$.

Here is another well-known example from the literature that is hard to crack in a purely representational setting (a piece of evidence against the claim, by the way, that 'or' in natural language is externally static):

John owns a motorbike or a car. It is in the garage. (8.2)

Again, in the tableau setting there is no problem: the tableau for (8.2) will have two branches, and both of the branches will contain a suitable antecedent for *it*.

*Computation of Answer Substitutions* The following example illustrates how the tableau calculus can be used to compute answer substitutions for a query.

$$\dfrac{\dfrac{\dfrac{x<3, x\doteq 5, [5/x]}{x<3, x\doteq 5} \quad \dfrac{x<3, x\doteq 2, [2/x]}{x<3, x\doteq 2}}{x<3, x\doteq 5 \cup x\doteq 2}}{x<3; x\doteq 5 \cup x\doteq 2}$$

A combination with model checking can be used to get rid of the left branch. Adding the relevant axioms for $<$ would achieve the same. See the next example.

*Reasoning about '<'* Assume that $1, 2, 3, \ldots$ are shorthand for $s0, ss0, sss0, \ldots$. We derive a contradiction from the assumption that $4 < 2$ together with two axioms for $<$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{4<2, \neg 4<2, \ldots} \quad \dfrac{\dfrac{}{4<2, 3<1, \neg 3<1} \quad \dfrac{\dfrac{}{4<2, 3<1, 2<0, \neg 2<0}}{4<2, 3<1, 2<0, \neg\{[2/x]; x<0\}}}{4<2, 3<1, \neg\{[2/x, 0/y]; sx<sy; \neg x<y\}, \ldots}}{4<2, \neg\{[3/x, 1/y]; sx<sy; \neg x<y\}, \ldots}}{4<2, \neg\{\exists x; \exists y; sx<sy; \neg x<y\}, \neg\{\exists x; x<0\}}}{4<2; \neg\{\exists x; x<0\}; \neg\{\exists x; \exists y; sx<sy; \neg x<y\}}$$

*Computation of Answer Substitutions, with Variable Reuse*

$$\dfrac{\dfrac{\dfrac{\dfrac{x\doteq 0, 0\doteq y, x_1\doteq 2, [0/y, 2/x_1]}{x\doteq 0, 0\doteq y, [0/y]; x_1\doteq 2} \quad \dfrac{x\doteq 0, y\doteq 2, x_1\doteq 2, [2/y, 2/x_1]}{x\doteq 0, y\doteq 2, [2/y]; x_1\doteq 2}}{x\doteq 0, 0\doteq y, [0/x, 0/y]; \exists x; x\doteq 2} \quad \dfrac{}{x\doteq 0, y\doteq 2, [0/x, 2/y]; \exists x; x\doteq 2}}{x\doteq 0, [0/x]; x\doteq y \cup y\doteq 2; \exists x; x\doteq 2}}{x\doteq 0; x\doteq y \cup y\doteq 2; \exists x; x\doteq 2}$$

Note how the computed answer substitution stores the final value for $x$, under the renaming $x_1$. Because of the renaming, the database information for $x_1$ does not conflict with that for $x$.

*Closure by Suspended Equality* This example illustrates closure by means of an equality suspended in a leaf node. Note that $x_1, y_1, x_2$ serve as names for objects in the domain under construction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x_1\neq y_1, x_2\doteq x_1, x_2\doteq y_1}\ x_1\approx y_1 \text{ suspended}}{x_1\neq y_1, \neg\{[x_1/y]; x_2\neq y\}, \neg\{[y_1/y]; x_2\neq y\}}}{x_1\neq y_1, \neg\{\exists y; x_2\neq y\}}}{x_1\neq y_1; \exists x; \neg\{\exists y; x\neq y\}}}{\exists x; \exists y; x\neq y; \exists x; \neg\{\exists y; x\neq y\}}$$

The equality $x_1 \approx y_1$ is suspended in the leaf node because of the presence of $x_2 \doteq x_1, x_2 \doteq y_1$ at that node. This suspended equality contradicts $x_1 \neq y_1$, and closure of the branch and the tableau.

*Loop Invariant Checking* To check that $x = y!$ is a loop invariant for $y := y + 1; x := x * y$, assume it is not, and use the calculus to derive a contradiction with the definition of !. Note that $y := y + 1; x := x * y$ appears in our notation as $[y + 1/y]; [x * y/x]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y! * (y+1) \neq (y+1)!}}{[y+1/y, y! * (y+1)/x]; x \neq y!}}{[y!/x, y+1/y]; [x * y/x]; x \neq y!}}{[y!/x]; [y+1/y]; [x * y/x]; x \neq y!}}{x = y!; [y+1/y]; [x * y/x]; x \neq y!}$$

A more detailed account would of course have to use the DFOL definitions of $+$, $*$ and !. In any case, the example should make our point about the potential of our calculus for Hoare style reasoning.

*Loop Invariant Checking: Variation* Let us check that $x = y!$ is also a loop invariant for $x := x * (y + 1); y := y + 1$. Again, assume that it is not, and calculate as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y! * (y+1) \neq (y+1)!}}{[y! * (y+1)/x, y+1/y]; x \neq y!}}{[y! * (y+1)/x]; [y+1/y]; x \neq y!}}{[y!/x]; [x * (y+1)/x]; [y+1/y]; x \neq y!}}{x = y!; [x * (y+1)/x]; [y+1/y]; x \neq y!}$$

*Postcondition Reasoning for 'If Then Else'* For another example of this, consider a loop through the following programming code:

$$i := i + 1; \text{if } x < a[i] \text{ then } x := a[i] \text{ else skip.} \tag{8.3}$$

Assume we know that before the loop $x$ is the maximum of array elements $a[0]$ through $a[i]$. Then our calculus allows us to derive a characterization of the value of $x$ at the end of the loop. Note that the loop code appears in DFOL under the following guise:

$$[i + 1/i]; (x < a[i]; [a[i]/x] \cup \neg x < a[i]).$$

The situation of $x$ at the start of the loop can be given by an identity $x = m_i^0$, where $m$ is a two-placed function. To get a characterization of $x$ at the end, we just put $X = x$ ($X$ a constant) at the end, and see what we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{m_i^0 < a[i+1], X = a[i+1], [i+1/i, a[i+1]/x]}{m_i^0 < a[i+1], [i+1/i, a[i+1]/x]; X = x}}{m_i^0 < a[i+1], [m_i^0/x, i+1/i]; [a[i]/x]; X = x}}{[m_i^0/x, i+1/i]; x < a[i]; [a[i]/x]; X = x} \quad \dfrac{\dfrac{\neg m_i^0 < a[i+1], X = m_i^0, [m_i^0/x, i+1/i]}{\neg m_i^0 < a[i+1], [m_i^0/x, i+1/i]; X = x}}{[m_i^0/x, i+1/i]; \neg x < a[i]; X = x}}{[m_i^0/x, i+1/i]; x < a[i]; [a[i]/x]; X = x \cup [m_i^0/x, i+1/i]; \neg x < a[i]; X = x}}{[m_i^0/x, i+1/i]; (x < a[i]; [a[i]/x] \cup \neg x < a[i]); X = x}}{[m_i^0/x]; [i+1/i]; (x < a[i]; [a[i]/x] \cup \neg x < a[i]); X = x}}{x = m_i^0; [i+1/i]; (x < a[i]; [a[i]/x] \cup \neg x < a[i]); X = x}$$

What the leaf nodes tell us is that in any case, $X$ is the maximum of $a[0], .., a[i+1]$, and this maximum gets computed in $x$.

*Strongest Postcondition Generation* Looking at the last example a bit more systematically, we see that $x = m_i^0$ is a precondition, and the two open nodes of the tableau contain sets of literals specifying the conditions leading to this node. In general, we can read off from the open nodes a propositional formula in disjunctive normal form that specifies the output in terms of the values of the input variables. In the present example, this is the formula:

$$(m_i^0 < a[i+1] \wedge X = a[i+1]) \vee (\neg m_i^0 < a[i+1] \wedge X = m_i^0),$$

This formula is in fact the strongest postcondition on the output value of $x$, given program (8.3) and precondition $x = m_i^0$, in terms of the input value of $i$. The output value of $x$ is given by $X$. Note that the constant $X$ plays the role of what is called a 'shadow variable' in Hoare style precondition/postcondition reasoning (see, e.g., Gordon [Gor88]).

The recipe for automatic generation of the strongest postcondition on a list of variables $x_1, \ldots, x_n$ of DFOL program $P$ under precondition $\phi$ is just this: put the calculus to work on

$$\phi; P; X_1 = x_1; \ldots; X_n = x_n,$$

and read off a formula in disjunctive normal form from the open tableau branches by collecting all the literals in $x_1, \ldots, x_n, X_1, \ldots, X_n$.

## 9. Completeness

Completeness for this calculus can be proved by a variation on completeness proofs for tableau calculi in classical FOL. First we define *trace sets* for DFOL as an analogue to Hintikka sets for FOL. A trace set is a set of DFOL formulas satisfying the closure conditions that can be read off from the tableau rules. Trace sets can be viewed as blow-by-blow accounts of particular consistent DFOL computation paths (i.e., paths that do not close). If the DFOL language has variables $V$ then the trace sets for the language will have variables $V \times \mathbb{N}$. In other words, in the traces, we are allowed to make as many copies of the variables in $V$ as we like. We will refer to a copy $\langle v, i \rangle$ as $v_i$.

**Definition 34** *A set $\Psi$ of $\mathcal{L}$ formulas, where $\mathcal{L}$ is a DFOL language over variable set $V \times \mathbb{N}$ is an $\mathcal{L}$* **trace set** *if the following hold:*

1. *For all substitution representations $\sigma$ it holds that $\neg\{\sigma\} \notin \Psi$.*
2. *If $\bar{t} \approx \bar{t}'$ is a list of equalities suspended in $\Psi$, then $P\bar{t}$ and $\neg P\bar{t}'$ do not both belong to $\Psi$. If $t \approx t'$ is an equality suspended in $\Psi$, then $t \neq t'$ does not belong to $\Psi$.*
3. *If $\sigma; \rho; \phi \in \Psi$ then $\sigma \circ \rho; \phi \in \Psi$, if $\{\sigma; \rho; \phi\} \in \Psi$ then $\{\sigma \circ \rho; \phi\} \in \Psi$, and if $\neg\{\sigma; \rho; \phi\} \in \Psi$ then $\neg\{\sigma \circ \rho; \phi\} \in \Psi$.*
4. *If $\sigma; P\bar{t}; \phi \in \Psi$, then $P\sigma\bar{t} \in \Psi$ and $\sigma; \phi \in \Psi$. If $\{\sigma; P\bar{t}; \phi\} \in \Psi$, then $P\sigma\bar{t} \in \Psi$ and $\{\sigma; \phi\} \in \Psi$.*

5. *If* $\neg\{\sigma; P\bar{t}; \phi\} \in \Psi$, *then either* $\neg P\sigma\bar{t} \in \Psi$ *or* $\neg\{\sigma; \phi\} \in \Psi$.

6. *If* $\sigma; t_1 \doteq t_2; \phi \in \Psi$ *and* $\sigma t_1 = v$, *then* $\sigma t_1 \doteq \sigma t_2 \in \Psi$ *and* $\sigma \circ [\sigma t_2/v]; \phi \in \Psi$. *If* $\{\sigma; t_1 \doteq t_2; \phi\} \in \Psi$ *and* $\sigma t_1 = v$, *then* $\sigma t_1 \doteq \sigma t_2 \in \Psi$ *and* $\{\sigma \circ [\sigma t_2/v]; \phi\} \in \Psi$. *Similarly for the case where* $\sigma t_2 = v$, *and the case where neither of* $\sigma t_1$, $\sigma t_2$ *is a variable.*

7. *If* $\neg\{\sigma; t_1 \doteq t_2; \phi\} \in \Psi$, *then either* $\neg\sigma t_1 \doteq \sigma t_2 \in \Psi$ *or* $\neg\{\sigma; \phi\} \in \Psi$.

8. *If* $\sigma; \exists v; \phi \in \Psi$ *then there is some* $v'$ *with* $\sigma; \phi^v_{v'} \in \Psi$. *If* $\{\sigma; \exists v; \phi\} \in \Psi$ *then there is some* $v'$ *with* $\{\sigma; \phi^v_{v'}\} \in \Psi$.

9. *If* $\neg\{\sigma; \exists v; \phi\} \in \Psi$, *then* $\neg\{\sigma \circ [t/v]; \phi\} \in \Psi$, *for every* $\mathcal{L}$ *term* $t$.

10. *If* $\sigma; (\phi_1 \cup \phi_2)\phi_3 \in \Psi$, *then either* $\sigma; \phi_1\phi_3 \in \Psi$ *or* $\sigma; \phi_2\phi_3 \in \Psi$. *If* $\{\sigma; (\phi_1 \cup \phi_2); \phi_3\} \in \Psi$, *then either* $\{\sigma; \phi_1; \phi_3\} \in \Psi$ *or* $\{\sigma; \phi_2; \phi_3\} \in \Psi$.

11. *If* $\neg\{\sigma; (\phi_1 \cup \phi_2); \phi_3\} \in \Psi$, *then both* $\neg\{\sigma; \phi_1; \phi_3\} \in \Psi$ *and* $\neg\{\sigma; \phi_2; \phi_3\} \in \Psi$.

12. *If* $\neg\{\sigma; \neg\{\phi_1\}; \phi_2\} \in \Psi$, *then either* $\{\sigma; \phi_1\} \in \Psi$ *or* $\neg\{\sigma; \phi_2\} \in \Psi$.

13. *If* $\sigma; \{\phi_1\}; \phi_2 \in \Psi$, *then both* $\{\sigma; \phi_1\} \in \Psi$ *and* $\sigma; \phi_2 \in \Psi$. *If* $\sigma; \neg\{\phi_1\}; \phi_2 \in \Psi$, *then both* $\neg\{\sigma; \phi_1\} \in \Psi$ *and* $\sigma; \phi_2 \in \Psi$.

14. *If* $\{\sigma; \{\phi_1\}; \phi_2\} \in \Psi$, *then* $\{\sigma; \phi_1\} \in \Psi$ *and* $\{\sigma; \phi_2\} \in \Psi$. *If* $\{\sigma; \neg\{\phi_1\}; \phi_2\} \in \Psi$, *then* $\neg\{\sigma; \phi_1\} \in \Psi$ *and* $\{\sigma; \phi_2\} \in \Psi$.

Next we prove the Trace Lemma:

**Lemma 35 (Trace Lemma)** *Every* $\mathcal{L}$ *trace set* $\Psi$ *over a set of terms* $U$ *is satisfiable in a domain* $U_{\equiv}$, *where* $\equiv$ *is the equivalence given by* $t \equiv t'$ *iff* $t \approx t'$ *is suspended in* $\Psi$.

**Proof.** First show that $\equiv$ is indeed an equivalence relation, so that $U_{\equiv}$ is well-defined. Next, let $[t]$ be the equivalence class of $t$ under $\equiv$. Define an interpretation over the domain $U_{\equiv}$, by means of: $I(f)[t_1]\cdots[t_n] := [ft_1\cdots t_n]$, $\langle[t_1]\cdots[t_n]\rangle \in I(P)$ iff $Pt'_1\cdots t'_n \in \Psi$, for some $t'_1, \ldots, t'_n$ with $t_1 \approx t'_1, \ldots, t_n \approx t'_n$ suspended in $\Psi$. Clearly, this model $\mathcal{M}$ is well-defined, by the properties of $\Psi$. Finally, define a valuation $s$ for $\mathcal{M}$ by means of $s(v) = [v]$, and establish that in this model $\mathcal{M}$ every member of $\Psi$ is true under $s$, in the following sense: for every $\phi \in \Psi$ there is a valuation $u$ with ${}_s[\![\phi]\!]^{\mathcal{M}}_u$. This check uses induction on the structure of the members of $\Psi$. □

To employ the lemma, we need the notion of a fair DFOL tableau: a tableau with the property that if it runs on indefinitely, still all the infinite open branches correspond to trace sets.

**Definition 36** *An infinite DFOL tableau branch is fair if all* $\neg\{\exists\cdots\}$ *obligations for new individuals* $v_{new}$ *are met along the branch. A DFOL tableau is fair if all of its infinite branches are fair. A fair DFOL tableau is finished if none of its finite open branches can be extended any further by means of a rule application.*

**Theorem 37** *Any open branch in a finished fair DFOL tableau corresponds to a trace set.*

**Proof.** Immediate from the definition of a trace set. The requirement "If $\neg\{\sigma; \exists v; \phi\} \in \Psi$, then $\neg\{\sigma \circ [t/v]; \phi\} \in \Psi$, for every $\mathcal{L}$ term $t$" is satisfied, for the language of the trace set, by the fact that the tableau is fair. □

The notion of simultaneous satisfiability of FOL applies to trace sets.

**Theorem 38** *Any open branch in a finished fair DFOL tableau is simultaneously satisfiable.*

**Proof.** Immediate from Lemma 35 and Theorem 37. □

**Theorem 39 (Completeness)** *For all $\phi, \psi \in \mathcal{L}$: if $\phi \models \psi$ then the tableau for $\phi; \neg\{\psi\}$ closes.*

**Proof.** Assume the tableau for $\phi; \neg\{\psi\}$ remains open. To make sure that this is not an accident of the order in which the rules were applied, assume also that the tableau is fair. Then there is at least one open branch which gives us a trace set. Use the trace set to construct a canonical model $\mathcal{M}$ and a canonical valuation $s$. This model will make $\phi; \neg\{\psi\}$ true, so we have: for no $s'$ with ${}_s[\![\phi]\!]^{\mathcal{M}}_{s'}$ is there a $u$ with ${}_{s'}[\![\psi]\!]^{\mathcal{M}}_{u}$. This establishes $\phi \not\models \psi$. □

**Theorem 40 (Computation Theorem)** *Any finite open branch of a fair DFOL tableau for $\phi$ yields a computed answer substitution $\sigma$ for $\phi$, in the sense that $\sigma$ satisfies ${}_s[\![\phi]\!]^{\mathcal{M}}_{s_\sigma}$, where $\mathcal{M}$ is the canonical model and $s$ the canonical valuation for that branch.*

**Proof.** Call a substitution $\sigma$ unblocked in $\Psi$ if there is some $\psi$ with $\sigma; \psi \in \Psi$. Check that the tableau rules that work on blocked or negated formulas never yield unblocked substitutions. Check that the nonsplitting tableau rules that work on unblocked substitutions always yield unblocked substitutions, and that the splitting tableau rules that work on unblocked substitutions always yield precisely one unblocked substitution. It follows that every tableau node has at most one formula of the form $\sigma; \psi$, with $\sigma \neq []$. A node of the form $\sigma; \psi$, with $\psi$ nonempty, cannot appear in an open end node, because it can always be further decomposed. Thus any open end node contains precisely one computed substitution $\sigma$, which can be thought of as the result of pulling the initial substitution [] through $\phi$. It can be proved by induction on the length of the tableau branch that ${}_s[\![\phi]\!]^{\mathcal{M}}_{s_\sigma}$, for $\mathcal{M}$ the canonical model and $s$ the canonical valuation for that branch. □

Note that since DFOL tableaux may have an infinite number of open branches, there are cases where DFOL computation yields an infinite number of solutions.

## 10. Variation: Using the Calculus with a Fixed Model

Computing with respect to a fixed model is but a slight variation on the general scheme. The technique of using tableau rules for model checking is well known. Assume that a model $\mathcal{M} = (D, I)$ is given. Then instead of storing ground predicates $P\sigma\bar{t}$ (ground equalities $\sigma t_1 \doteq \sigma t_2$), we check the model for $\mathcal{M} \models P\sigma\bar{t}$ (for $[\![\sigma t_1]\!]^{\mathcal{M}} = [\![\sigma t_2]\!]^{\mathcal{M}}$), and close the branch if the test fails, continue otherwise. Similarly, instead of storing ground predicates $P\sigma\bar{t}$ (ground equalities $\sigma t_1 \doteq \sigma t_2$) under negation, we check the model for $\mathcal{M} \not\models P\sigma\bar{t}$ (for $[\![\sigma t_1]\!]^{\mathcal{M}} \neq [\![\sigma t_2]\!]^{\mathcal{M}}$), and close the branch if the test fails, continue otherwise.

## 11. Adding Iteration

Let $\mathcal{L}^*$ be the language that results form extending $\mathcal{L}$ with formulas of the form $\phi^*$. The intended relational meaning of $\phi^*$ is that $\phi$ gets executed a finite ($\geq 0$) number of times. This extension makes $\mathcal{L}^*$ into a full fledged programming language, with its assertion language built in for good measure.

The semantic clause for $\phi^*$ runs as follows:

$$ {}_s[\![\phi^*]\!]^{\mathcal{M}}_u \quad \text{iff} \quad \begin{array}{l} \text{either } s = u \\ \text{or } \exists s_1, \ldots, s_n (n \geq 1) \text{ with } {}_s[\![\phi]\!]^{\mathcal{M}}_{s_1}, \ldots, {}_{s_n}[\![\phi]\!]^{\mathcal{M}}_u. \end{array} $$

It is easy to see that it follows from this definition that:

$$ {}_s[\![\phi^*]\!]^{\mathcal{M}}_u \text{ iff either } s = u \text{ or } \exists s_1 \text{ with } {}_s[\![\phi]\!]^{\mathcal{M}}_{s_1} \text{ and } {}_{s_1}[\![\phi^*]\!]^{\mathcal{M}}_u. \tag{11.1} $$

Note, however, that (11.1) is not equivalent to the definition of ${}_s[\![\phi^*]\!]^{\mathcal{M}}_u$, for (11.1) does not rule out infinite $\phi$ paths.

It is useful to introduce a new abbreviation $\phi^n$, given by: $\phi^0 := \top$, $\phi^{n+1} := \phi; \phi^n$. Now $\phi^*$ is equivalent to 'for some $n \in \mathbb{N} : \phi^n$'.

What we will do in our calculus for DFOL$^*$ is take (11.1) as the cue to the star rules. This will allow star computations to loop, which is all right, given that we extend our notion of closure to 'closure in the limit' (see below).

Figure 2: The Calculus for DFOL$^*$

All rules of the DFOL calculus, plus ...

| | |
|---|---|
| star + | $\dfrac{\phi; \chi + \Phi \qquad \phi; \psi; \psi^*; \chi + \Phi}{\phi; \psi^*; \chi + \Phi}$ $\qquad$ $\dfrac{\{\phi; \chi\} + \Phi \qquad \{\phi; \psi; \psi^*; \chi\} + \Phi}{\{\phi; \psi^*; \chi\} + \Phi}$ |
| star − | $\dfrac{\neg\{\phi; \chi\} + \neg\{\phi; \psi; \psi^*; \chi\} + \Phi}{\neg\{\phi; \psi^*; \chi\} + \Phi}$ |

The calculus for DFOL$^*$ is given in Figure 2. It consists of the calculus for DFOL plus three rules for star. The star rules are inflationary, so they may lead to infinite tableaux, just like the rule for quantication under negation.

To deal with the new source of infinity in tableau development, we need a modification of our notion of tableau closure. We allow closure in the limit, as follows.

**Definition 41** *An infinite tableau branch closes in the limit if it contains an infinite star development, i.e., an infinite number of star applications to* the same star formula, *where $\psi^*$ in the residual $\phi; \psi; \psi^*; \chi$ of the star rule application to $\phi; \psi^*; \chi$ counts as* the same star formula *(and similarly for the other star rules).*

To see that the first of the star rules is sound, assume that ${}_s[\![\phi; \psi^*; \chi]\!]^{\mathcal{M}}_u$. Then there is an $s'$ with ${}_s[\![\phi]\!]^{\mathcal{M}}_{s'}$ and ${}_{s'}[\![\psi^*; \chi]\!]^{\mathcal{M}}_u$. Then, by (11.1), either ${}_{s'}[\![\chi]\!]^{\mathcal{M}}_u$ or there is a $s_1$ with ${}_{s'}[\![\psi]\!]^{\mathcal{M}}_{s_1}$ and

${}_{s_1}[\![\phi_1^*;\chi]\!]_u^{\mathcal{M}}$. It follows that if $\Phi+\phi;\psi^*;\chi$ is consistent, then either $\Phi+\phi;\chi$ or $\Phi+\phi;\psi;\psi^*;\chi$ is consistent. The reasoning for star under block is similar.

For the rule for star under negation, assume that ${}_s[\![\neg\{\phi;\psi^*;\chi\}]\!]_u^{\mathcal{M}}$. Then $s=u$ and there is no $s'$ with ${}_s[\![\phi;\psi^*;\chi]\!]_{s'}^{\mathcal{M}}$. By (11.1), this means that there is no $s'$ with ${}_s[\![\phi;\chi]\!]_{s'}^{\mathcal{M}}$ and no $s'$ with ${}_s[\![\phi;\psi;\psi^*;\chi]\!]_{s'}^{\mathcal{M}}$. Thus, if $\Phi+\neg\{\phi;\psi^*;\chi\}$ is consistent, then $\Phi+\neg\{\phi;\chi\}+\neg\{\phi;\psi;\psi^*;\chi\}$ is also consistent.

*Infinite Closure in the Limit* We will give an example of an infinite star development. Consider formula (11.2):

$$\neg\exists w\neg\{\exists v;v=0;(v\neq w;[v+1/v])^*;v=w\}. \tag{11.2}$$

What (11.2) says is that there is no object $w$ that cannot be reached in a finite number of steps from $v=0$, or in other words that the successor relation $v\mapsto v+1$, considered as a graph, is wellfounded. This is the Peano induction axiom: it characterizes the natural numbers up to isomorphism. What it says is that any set $A$ that contains 0 and is closed under successor contains all the natural numbers. The fact that Peano induction is expressible as an $\mathcal{L}^*$ formula is evidence that $\mathcal{L}^*$ has greater expressive power than FOL. In FOL no single formula can express Peano induction: no formula can distinguish the standard model $(\mathbb{N},s)$ from the non-standard models. In a non-standard model of the natural numbers it may take an infinite number of $s$-steps to get from one natural number $n$ to a larger number $m$.

The expressive power of $\mathcal{L}^*$ is the same as that of quantified dynamic logic ([Pra76, Gol87]). Arithmetical truth is undecidable, so there can be no finitary refutation system for $\mathcal{L}^*$. The finitary tableau system for $\mathcal{L}$ is evidence for the fact that DFOL validity is recursively enumerable: all non-validities are detected by a finite tableau refutation. This property is lost in the case of $\mathcal{L}^*$: the language is just too expressive to admit of finitary tableau refutations.

Therefore, some tableau refutations must be infinitary, and the tableau development for the negation of (11.2) is a case in point. Let us see what happens if we attempt to refute the negation of (11.2). A successful refutation will identify the natural numbers up to isomorphism.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{2\neq w,\neg\{[3/v];(v\neq w;[v+1/v])^*;v\doteq w\}}}{1\neq w,\neg\{[2/v];(v\neq w;[v+1/v])^*;v\doteq w\}}}{0\neq w,\neg\{[1/v];(v\neq w;[v+1/v])^*;v\doteq w\}}}{\neg\{[0/v];(v\neq w;[v+1/v])^*;v\doteq w\}}}{\exists w\neg\{\exists v;v\doteq 0;(v\neq w;[v+1/v])^*;v\doteq w\}}$$

This is indeed a successful refutation, for the tree closes in the limit. But the refutation tree is infinite: it takes an infinite amount of time to do all the checks.

**Theorem 42 (Soundness Theorem for $\mathcal{L}^*$)** *The calculus for DFOL$^*$ is sound:*

*For all $\phi,\psi\in\mathcal{L}^*$: if the tableau for $\phi;\neg\{\psi\}$ closes then $\phi\models\psi$.*

The modified tableau method does not always give finite refutations. Still, it is a very useful reasoning tool, more powerful than Hoare reasoning, and more practical than the infinitary calculus for quantified dynamic logic developed in [Gol82, Gol87]. Unlike Hoare logic, Goldblatt's infinitary calculus for QDL was never designed for practical use, and was, as far as we know, never used for any practical purposes.

*Precondition/postcondition Reasoning* For a further example of reasoning with the calculus, consider formula (11.3). This gives an $\mathcal{L}^*$ version of Euclid's GCD algorithm.

$$(x \neq y; (x > y; [x - y/x] \cup y > x; [y - x/y]))^*; x \doteq y. \tag{11.3}$$

To do automated precondition-postcondition reasoning on this, we must find a trivial correctness statement. Even if we don't know what $\gcd(x, y)$ is, we know that its value should not change during the program. So putting $\gcd(x, y)$ equal to some arbitrary value and see what happens would seem to be a good start. We will use the correctness statement $z \doteq \gcd(x, y)$. The statement that the result gets computed in $x$ can then take the form $z \doteq x$. The program with these trivial correctness statements included becomes:

$$\begin{array}{l} z \doteq \gcd(x, y); \\ (x \neq y; (x > y; [x - y/x]; z \doteq \gcd(x, y) \cup y > x; [y - x/y]; z \doteq \gcd(x, y)))^*; \\ x \doteq y; z \doteq x. \end{array} \tag{11.4}$$

We can now put the calculus to work. Abbreviating $(x \neq y; (x > y; [x - y/x]; z \doteq \gcd(x, y) \cup y > x; [y - x/y]; z \doteq \gcd(x, y)))^*$ as $A^*$, we get:

$$\dfrac{\dfrac{x \doteq y, \gcd(x, y) \doteq x}{[\gcd(x, y)/z]; x \doteq y; z \doteq x} \qquad \dfrac{x > y, \gcd(x, y) \doteq \gcd(x - y, y), \phi \qquad y > x, \gcd(x, y) \doteq \gcd(x, y - x), \psi}{[\gcd(x, y)/z]; A; A^*; x \doteq y; z \doteq x}}{z \doteq \gcd(x, y); A^*; x \doteq y; z \doteq x}$$

Here $\phi \doteq [\gcd(x, y)/z, x - y/x]; A^*; x \doteq y; z \doteq x$ and $\psi \doteq [\gcd(x, y)/z, y - x/y]; A^*; x \doteq y; z \doteq x$. The second split is caused by an application of the rule for $\cup$.

By the soundness of the calculus any model satisfying the annotated program (11.4) will satisfy one of the branches. This shows that if the program succeeds (computes an answer), the following disjunction will be true:

$$\begin{array}{ll} & (x \doteq y \wedge \gcd(x, y) \doteq x) \\ \vee & (x > y \wedge \gcd(x, y) \doteq \gcd(x - y, y) \wedge \phi) \\ \vee & (y > x \wedge \gcd(x, y) \doteq \gcd(x, y - x) \wedge \psi) \end{array} \tag{11.5}$$

From this it follows that the following weaker disjunction is also true:

$$\begin{array}{ll} & (x \doteq y \wedge \gcd(x, y) \doteq x) \\ \vee & (x > y \wedge \gcd(x, y) \doteq \gcd(x - y, y)) \\ \vee & (y > x \wedge \gcd(x, y) \doteq \gcd(x, y - x)) \end{array} \tag{11.6}$$

Note that (11.6) looks remarkably like a functional program for GCD.

## 12. Completeness for DFOL*

The method of trace sets for proving completeness from Section 9 still applies. Trace sets for DFOL* will have to satisfy the following extra conditions:

15. If $\phi;\psi^*;\chi \in \Psi$, then either $\phi;\chi \in \Psi$ or $\phi;\psi;\psi^*;\chi \in \Psi$. If $\{\phi;\psi^*;\chi\} \in \Psi$, then either $\{\phi;\chi\} \in \Psi$ or $\{\phi;\psi;\psi^*;\chi\} \in \Psi$.

16. If $\phi;\psi^*;\chi \in \Psi$, then there is some $n \geq 0$ with $\phi;\psi^m;\chi \notin \Psi$ for all $m > n$. If $\{\phi;\psi^*;\chi\} \in \Psi$, then there is some $n \geq 0$ with $\{\phi;\psi^m;\chi\} \notin \Psi$ for all $m > n$.

17. For all $\phi, \psi, \chi \in \Psi$ it holds that $\neg\{\phi;\psi^*;\chi\} \notin \Psi$.

In order to preserve the correspondence between trace sets and open tableau branches, we must adapt the definition of a fair tableau.

**Definition 43** *An infinite DFOL* tableau branch is fair if*

1. *all $\neg\{\exists \cdots\}$ obligations for new individuals $v_{new}$ are met along the branch,*
2. *all $\phi;\psi^*;\chi$ and $\{\phi;\psi^*;\chi\}$ obligations are met along the branch.*

*A DFOL* tableau is fair if all of its infinite branches are fair. A fair DFOL* tableau is finished if none of its finite open branches can be extended any further by means of a rule application.*

We can again prove a trace lemma for DFOL*, in the same manner as before:

**Lemma 44 (Trace Lemma for DFOL*)** *Every $\mathcal{L}$ trace set $\Psi$ over a set of terms $U$ is satisfiable in a domain $U_{\equiv}$, where $\equiv$ is the equivalence given by $t \equiv t'$ iff $t \approx t'$ is suspended in $\Psi$.*

Again, open branches in finished fair DFOL* tableaux will correspond to trace sets, and we can satisfy these trace sets in canonical models. Note that requirements (16) and (17) are met thanks to our stipulation about closure in the limit. Finally, we get:

**Theorem 45 (Completeness for $\mathcal{L}^*$)** *For all $\phi, \psi \in \mathcal{L}^*$: if $\phi \models \psi$ then the tableau for $\phi;\neg\{\psi\}$ closes.*

So we have a complete logic for DFOL*, but of course it comes at a price: we may occasionally get in a refutation loop. However, as our tableau construction examples illustrate, this does hardly affect the usefulness of the calculus.

## 13. Related Work

*Comparison with other Calculi for DFOL and for DRT* The calculus of Van Eijck [Eij99] uses swap rules for moving quantifiers to the front of formulas. The key idea of the present calculus is entirely different: encode dynamic binding in explicit substititions and protect outside environments from dynamic side effects by means of block operations. In a sense, the present calculus offers a full account of the phenomenon of local variable use in DFOL.

Kohlhase [Koh00] gives a tableau calculus for DRT (Discourse Representation Theory, see [Kam81]) that has essentially the same scope as the [Eij99] calculus for DPL: the version of DRT disjunction that is treated is externally static, and the DRT analogue of $\cup$ is not treated.

The Kohlhase calculus follows an old DRT tradition in relying on an implicit translation to standard FOL: see [SE88] for an earlier example of this. Kohlhase motivates his calculus with the need for (minimal) model generation in dynamic NL semantics. In order to make his calculus generate minimal models, he replaces the rule for existential quantification by a 'scratchpaper' version (well-known from textbook treatments of tableau reasoning): first try out if you can avoid closure with a term already available at the node. If all these attempts result in closure, it does not follow from this that the information at the node is inconsistent, for it may just be that we have 'overburdened' the available terms with demands. So in this case, and only in this case, introduce a new individidual.

This 'exhaustion of existing terms' approach has the virtue that it generates 'small' models when they exist, whereas the more general procedure 'always introduce a new term' may generate infinite models where finite models exist. Note, however, that the strategy only makes sense for a signature without function symbols. In our format, it would suggest a quantification rule like the following:

$$\frac{\phi \circ [t_1/v]; \phi + \Phi \quad \ldots \quad \sigma \circ [t_n/v]; \phi + \Phi \quad \sigma; \phi^v_{v_{\mathrm{new}}} + \Phi}{\sigma; \exists v; \phi + \Phi}$$

where $t_1, \ldots, t_n$ are the finitely many available terms of the current tableau branch. Kohlhase discusses applications in NL processing, where it often makes sense to construct a minimal model for a text, and where the assumption of minimality can be used to facilitate issues of anaphora resolution and presupposition handling.

*Comparison with Apt and Bezem* Apt and Bezem present what can be viewed as an exciting new mix of tableau style reasoning and model checking for FOL. Our treatment of equality uses a generalization of a stratagem from their [AB99]: in the context of a partial variable map $\sigma$, they call $v \doteq t$ a $\sigma$ *assignment* if $v \notin dom(\sigma)$, and all variables occurring in $t$ are in $dom(\sigma)$. We generalize this on two counts:

- Because our computation results are substitutions (term maps) rather than maps to objects in the domain of some model, we allow computation of non-ground terms as values.
- Because our substitutions are total, in our calculus execution of $t_1 \doteq t_2$ atoms never gives rise to an error condition.

It should be noted for the record that the first of these points is addressed in [Apt00].

Apt and Bezem present their work as an underpinning for Alma-0, a language that infuses Modula style imperative programming with features from logic programming (see [ABPS98]). In a similar way, the present calculus provides logical underpinnings for Dynamo, a language for programming with an extension of DFOL. For a detailed comparison of Alma-0 and Dynamo we refer the reader to [Eij98].

*Comparison with tableau reasoning for FOL and for modal fragments of FOL* The present calculus for DFOL can be viewed as a more dynamic version of tableau style reasoning for FOL and for modal fragments of FOL. Instead of just checking for valid consequence and constructing counterexamples from open tableau branches, our open tableau branches yield computed answer substitutions as an extra. The connection with tableau reasoning for FOL is also evident in the proof method of our completeness theorems.

*Connection with WHILE, GCL* It is easy to give an explicit substitution semantics for WHILE, the favorite toy language of imperative programming from the textbooks (see e.g., [NN92]), or for GCL, the non-deterministic variation on this proposed by Dijkstra (see, e.g. [DS90]). DFOL is in fact quite closely related to these, and it is not hard to see that DFOL$^*$ has the same expressive power as GCL. Our tableau calculus for DFOL$^*$ can therefore be regarded as an execution engine *cum* reasoning engine for WHILE or GCL.

*Connection with PDL, QDL* There is also a close connection between DFOL$^*$ on one hand and propositional dynamic logic (PDL) and quantified dynamic logic (QDL) on the other. QDL is a language proposed in [Pra76] to analyze imperative programming, and PLD is its propositional version. See [Seg82, Par78] for complete axiomatisations of PDL, and [Gol87] for an excellent exposition of both PDL and QDL, and for a complete axiomatisation of QDL. In PDL/QDL, programs are treated as modalities and assertions about programs are formulas in which the programs occur as modal operators. Thus, if $\pi$ is a program, $\langle\pi\rangle\phi$ asserts that $\pi$ has a successful termination ending in a state satisfying $\phi$. This cannot be expressed without further ado in Hoare logic, by the way.

The main difference between DFOL$^*$ and PDL/QDL is that in DFOL$^*$ the distinction between formulas and programs is abolished. Everything is a program, and assertions about programs are test programs that are executed along the way, but with their dynamic effects blocked. To express that $\pi$ has a successful termination ending in a $\phi$ state, we can just say $\{\pi;\phi\}$. To check whether $\pi$ has a successful termination ending in a $\phi$ state, try to refute the statement by constructing a tableau for $\neg\{\pi;\phi\}$.

To illustrate the connection with QDL and PDL, consider MIX, the first of the two PDL axioms for $*$:

$$[\pi^*]\phi \to \phi \wedge [\pi][\pi^*]\phi. \tag{13.1}$$

Writing this with $\langle\pi\rangle$, $\neg$, $\wedge$, $\vee$, and replacing $\neg\phi$ by $\phi$, we get:

$$\neg(\neg\langle\pi^*\rangle\phi \wedge (\phi \vee \langle\pi\rangle\langle\pi^*\rangle\phi)). \tag{13.2}$$

This has the following DFOL$^*$ counterpart:

$$\neg\{\neg\{\pi^*;\phi\};(\phi \cup \{\pi;\pi^*;\phi\})\}. \tag{13.3}$$

For a refutation proof of (13.3), we leave out the outermost negation.

$$\dfrac{\dfrac{\dfrac{\overline{\phi} \quad \overline{\{\pi;\pi^*;\phi\}}}{\neg\phi, \neg\{\pi;\pi^*;\phi\}, (\phi \cup \{\pi;\pi^*;\phi\})}}{\neg\{\pi^*;\phi\}, (\phi \cup \{\pi;\pi^*;\phi\})}}{\neg\{\pi^*;\phi\};(\phi \cup \{\pi;\pi^*;\phi\})}$$

The tableau closes, so we have proved that (13.3) is a DFOL$^*$ theorem (and thus, a DFOL$^*$ validity).

We will also derive the validity of the DFOL$^*$ counterpart to IND, the other PDL axiom for $*$:

$$(\phi \wedge [\pi^*](\phi \to [\pi]\phi)) \to [\pi^*]\phi. \tag{13.4}$$

Equivalently, this can be written with only $\langle\pi\rangle, \neg, \wedge, \vee$, as follows:

$$\neg(\phi \wedge \neg\langle\pi^*\rangle(\phi \wedge \langle\pi\rangle\neg\phi) \wedge \langle\pi^*\rangle\neg\phi). \tag{13.5}$$

The DFOL$^*$ counterpart of (13.5) is:

$$\neg\{\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \pi^*; \neg\phi\}. \tag{13.6}$$

We will give a refutation proof of (13.6) in two stages. First, we show that (13.7) can be refuted for any $n \geq 0$, and next, we use this for the proof of (13.6).

$$\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \pi^n; \neg\phi. \tag{13.7}$$

Here is the case of (13.7) with $n = 0$:

$$\dfrac{\dfrac{}{\phi, \neg\{\pi^*; \phi; \pi; \neg\phi\}, \neg\phi}}{\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \neg\phi}$$

Bearing in mind that that $\pi$ is a dynamic action and $\phi$ is a test, we can apply the rule of Negation Splitting (Theorem 33) to formulas of the form $\neg\{\pi^n; \phi; \pi; \neg\phi\}$, as follows:

$$\dfrac{\{\pi^n; \neg\phi\} \quad \neg\{\pi^{n+1}; \neg\phi\}}{\neg\{\pi^n; \phi; \pi; \neg\phi\}}$$

Note that $\neg\{\pi^n; \phi; \pi; \neg\phi\}$ can be derived from $\neg\{\pi^*; \phi; \pi; \neg\phi\}$ by $n$ applications of the $*-$ rule. Using this, we get the following refutation tableau for the case of (13.7) with $n = k+1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\pi^k; \neg\phi\}} \quad \dfrac{}{\neg\{\pi^{k+1}; \neg\phi\}}}{\neg\{\pi^k; \phi; \pi; \neg\phi\}}}{\phi, \neg\{\pi^*; \phi; \pi; \neg\phi\}, \pi^{k+1}; \neg\phi}}{\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \pi^{k+1}; \neg\phi}$$

The lefthand branch closes because of the refutation of $\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \pi^k; \neg\phi$, which is given by the induction hypothesis.

Next, use these refutations of $\neg\phi$, $\pi; \neg\phi$, $\pi^2; \neg\phi$, ..., to prove (13.6) by means of a refutation in the limit, as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\phi} \quad \dfrac{\dfrac{}{\pi; \neg\phi} \quad \dfrac{\dfrac{}{\pi^2; \neg\phi} \quad \dfrac{\vdots}{\pi^3; \pi^*; \neg\phi}}{\pi^2; \pi^*; \neg\phi}}{\pi; \pi^*; \neg\phi}}{\phi, \neg\{\pi^*; \phi; \pi; \neg\phi\}, \pi^*; \neg\phi}}{\phi; \neg\{\pi^*; \phi; \pi; \neg\phi\}; \pi^*; \neg\phi}$$

This closed tableau establishes (13.6) as a DFOL* theorem. That closure in the limit is needed to establish the DFOL* induction principle is not surprising. The DFOL * rules express that * computes a fixpoint, while the fact that this fixpoint is a *least* fixpoint is captured by the stipulation about closure in the limit. The induction principle (13.6) hinges on the fact that * computes a least fixpoint.

Goldblatt [Gol82, Gol87] develops an infinitary proof system for QDL with the following key rule of inference:

$$\text{If } \phi \to [\pi_1; \pi_2^n]\psi \text{ is a theorem for every } n \in \mathbb{N}, \text{ then } \phi \to [\pi_1; \pi_2^*]\psi \text{ is a theorem.} \quad (13.8)$$

To see how this is related to the present calculus, assume that one attempts to refute $\phi \to [\pi_1; \pi_2^*]\psi$, or rather, its DFOL* counterpart $\neg\{\phi; \pi_1; \pi_2^*; \neg\psi\}$, on the assumption that for any $n \in \mathbb{N}$ there exists a refutation for $\neg\{\phi; \pi_1; \pi_2^n; \neg\psi\}$. Again, for a refutation attempt we can leave out the outermost negation.

$$\dfrac{\dfrac{}{\phi; \pi_1; \neg\psi} \quad \dfrac{\dfrac{}{\phi; \pi_1; \pi_2; \neg\psi} \quad \dfrac{\dfrac{}{\phi; \pi_1; \pi_2; \pi_2; \neg\psi} \quad \dfrac{\vdots}{\phi; \pi_1; \pi_2; \pi_2; \pi_2^*; \neg\psi}}{\phi; \pi_1; \pi_2; \pi_2; \pi_2^*; \neg\psi}}{\phi; \pi_1; \pi_2; \pi_2^*; \neg\psi}}{\phi; \pi_1; \pi_2^*; \neg\psi}$$

Note that we can close off the $\phi; \pi_1; \pi_2^n; \neg\psi$ branches by the assumption that there exist refutations for $\neg\{\phi, \pi_1; \pi_2^n; \neg\psi\}$, for every $n \in \mathbb{N}$. The whole tableau gives an infinite positive star development, and the infinite branch closes in the limit, so the tableau closes, thus establishing that in the DFOL* calculus valitity of $\neg\{\phi; \pi_1; \pi_2^*; \neg\psi\}$ follows from the fact that $\neg\{\phi; \pi_1; \pi_2^n; \neg\psi\}$ is valid for every $n \in \mathbb{N}$.

*Connection with item notation and explicit substitution in lambda calculus* Our treatment of explicit substitutions has the following connection with [KN95]. If $[t/v]; \phi$ is a formula starting with an explicit substitution $[t/v]$, we can view this as an abstraction $\lambda v.\phi$ applied to an argument $t$. In the item notation proposed in [KN95], this would in turn be written as $(t)[v]\phi$ (modulo some further rearrangements inside $\phi$). The main advantage of item notation in lambda calculus is that it allows for an easy formulation of a more powerful rule of lambda reduction, because of the convenient 'bracketing structure'. The resemblance suggests that our simultaneous explicit substitutions are related to a generalization of item notation, with constructions for simultaneous abstraction and application. $(t_1 \cdots t_n)[v_1 \cdots v_n]\phi$ would then be item notation for $[t_1/v_1, \ldots, t_n/v_n]; \phi$. We leave this connection for future exploration.

## 14. Conclusion

Starting out from an analysis of substitution in dynamic FOL, we have given a tableau calculus for reasoning with dynamic logic. The format for the calculus and the role of explicit substitutions for computing answers to queries were motivated by our search for logical underpinnings for programming with (an extension of) DFOL. The DFOL tableau calculus presented here constitutes the theoretical basis for *Dynamo*, a toy programming language based on DFOL. The versions of *Dynamo* implemented so far implement tableau reasoning for DFOL with respect to a fixed model: see [Eij98].

To find the answer to a query, given a formula $\phi$ considered as *Dynamo* program data, *Dynamo* essentially puts the tableau calculus to work on a formula $\phi$, all the while checking predicates with respect to the fixed model of the natural numbers, and storing values for variables from the inspection of equality statements. If the tableau closes, this means that $\phi$ is inconsistent (with the information obtained from testing on the natural numbers), and *Dynamo* reports 'false'. If the tableau remains open, *Dynamo* reports that $\phi$ is consistent (again with the information obtained from inspecting predicates on the natural numbers), and lists the computed substitutions for the output variables at the end of the open branches. But the *Dynamo* engine also works for general tableau reasoning, and for general queries. The literals collected along the open branches together with the explicit substitutions at the trail ends constitute the computed answers.

*Dynamo* can be viewed as a combined engine for program execution and reasoning. We are currently working on an new implementation of *Dynamo* that takes the insights reported above into account. The advantages of the combination of execution and reasoning embodied in *Dynamo* should be evident from our examples of strongest postcondition generation in Section 8. To our knowledge, this use of dynamic first order logic for analysing imperative programming by means of calculating trace sets is new. We claim that our calculus opens the road to a more intuitive way of reasoning about imperative programs.

Finally, since natural language semantics is a key application area of dynamic variations on first order logic, we expect that both the calculus itself and its implementation in the form of an improved execution mechanism for *Dynamo* also have a role to play in a truly computational semantics for natural language.

Acknowledgments

Thanks to Johan van Benthem, Balder ten Cate, Anne Kaldewaij, Fairouz Kamareddine, Michael Kohlhase, Maarten Marx, Joachim Niehren, Kees Vermeulen, Albert Visser, Joe Wells. Proposition (9) was triggered by a question from Krzysztof Apt.

# References


[AB99] K.R. Apt and M. Bezem. Formulas as programs. In K.R. Apt, V. Marek, M. Truszczyski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25 Years Perspective*, pages 75–107. Springer Verlag, 1999. Paper available as `http://xxx.lanl.gov/abs/cs.LO/9811017`.

[ABPS98] K.R. Apt, J. Brunekreef, V. Partington, and A. Schaerf. Alma-0: An imperative language that supports declarative programming. *ACM Toplas*, 1998.

[Apt97] K.R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[Apt00] K.R. Apt. A denotational semantics for first-order logic. In *Proc. of the Computational Logic Conference (CL2000)*, Notes in Artificial Intelligence 1861, pages 53–69. Springer, 2000.

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics (2nd ed.).* North-Holland, Amsterdam, 1984.

[BdM97] R. Bird and O. de Moor. *Algebra of Programming.* Prentice Hall, 1997.

[Ben86] J. van Benthem. Partiality and nonmonotonicity in classical logic. *Logique et Analyse*, 29, 1986.

[Ben96] J. van Benthem. *Exploring Logical Dynamics.* CSLI & Folli, 1996.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[BvE82] J. van Benthem and J. van Eijck. The dynamics of interpretation. *Journal of Semantics*, 1(1):3–20, 1982.

[Doe94] H.C. Doets. *From Logic to Logic Programming.* MIT Press, Cambridge, Massachusetts, 1994.

[DS90] E.W Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[Eij98] J. van Eijck. Programming with dynamic predicate logic. Technical Report CT-1998-06, ILLC, 1998. Available from `www.cwi.nl/~jve/dynamo`.

[Eij99] J. van Eijck. Axiomatising dynamic logics for anaphora. *Journal of Language and Computation*, 1:103–126, 1999.

[Fit90] M. Fitting. *First-order Logic and Automated Theorem Proving.* Springer Verlag, Berlin, 1990.

[Gol82] R. Goldblatt. *Axiomatising the Logic of Computer Programming.* Springer, 1982.

[Gol87] R. Goldblatt. *Logics of Time and Computation, Second Edition, Revised and Expanded*, volume 7 of *CSLI Lecture Notes.* CSLI, Stanford, 1992 (first edition 1987). Distributed by University of Chicago Press.

[Gor88] M.J.C. Gordon. *Programming language theory and its implementation: applicative and imperative paradigms.* Prentice Hall, 1988.

[GS91] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

[Han94] C. Hankin. *Lambda Calculi.* Clarendon Press, Oxford, 1994.

[Kam81] H. Kamp. A theory of truth and semantic representation. In J. Groenendijk et al., editors, *Formal Methods in the Study of Language.* Mathematisch Centrum, Amsterdam, 1981.

[KN95] F. Kamareddine and R. Nederpelt. Refining reduction in the lambda calculus. *Journal of Functional Programming*, 5:637–651, 1995.

[Koh00] M. Kohlhase. Model generation for Discoure Representation Theory. In *ECAI Proceedings*, 2000. Available from `http://www.ags.uni-sb.de/~kohlhase/`.

[NN92] H.R. Nielson and F. Nielson. *Semantics with Applications.* John Wiley and Sons, 1992.

[Par78] R. Parikh. The completeness of propositional dynamic logic. In *Mathematical Foundations of Computer Science 1978*, pages 403–415. Springer, 1978.

[Pra76] V. Pratt. Semantical considerations on Floyd–Hoare logic. *Proceedings 17th IEEE Symposium on Foundations of Computer Science*, pages 109–121, 1976.

[SE88] C. Sedogbo and M. Eytan. A tableau calculus for DRT. *Logique et Analyse*, 31:379–402, 1988.

[Seg82] K. Segerberg. A completeness theorem in the modal logic of programs. In T. Traczyck, editor, *Universal Algebra and Applications*, pages 36–46. Polish Science Publications, 1982.

[Smu68] R. Smullyan. *First-order logic.* Springer, Berlin, 1968.

[Ven95] Y. Venema. A modal logic of quantification and substitution. In L. Czirmaz, D.M. Gabbay, and M. de Rijke, editors, *Logic Colloquium '92*, Studies in Logic, Language and Computation, pages 293–309. CSLI and FOLLI, 1995.

[Vis98] A. Visser. Contexts in dynamic predicate logic. *Journal of Logic, Language and Information*, 7(1):21–52, 1998.

[Vis00] A. Visser. A note on substitution in dynamic semantics. Unpublished draft, Utrecht University, 2000.