Tableaux Theorem Proving for Dynamic First Order Logic

Jan van Eijck and Juan Heguiabehere and Breanndán Ó Nualláin

March, 2000

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 tableaux 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.

1 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 [13] for a presentation and Visser [18] for an in-depth analysis. A sound and complete sequent style calculus for DFOL (without choice) was presented in Van Eijck [9]. 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 Smulllyan [16] for a classical presentation, and Fitting [11] for connections with automated theorem proving).

For applications of DFOL to programming, the presence of the choice operation $\cup$ in the language is crucial, while it can be (and has been) argued that for applications of DFOL to natural language semantics it is not. In fact, the presentation of dynamic predicate logic in [13] does not cover $\cup$, and neither does the calculus of [9]. The present calculus deals with DFOL including choice.

A very convenient extension that we immediately add to DFOL is the extension with (representations of) simultaneous substitutions. It is well known that substitutions are definable in DFOL. Still we will consider them as semantic objects in their own right, in the spirit of Venema [17], where substitutions are studied as modal operators.

Let a signature for FOL be given. We will assume that the signature has individual constants but no function constants. Let V be the set of variables and T the set of terms. Then a substitution σ is a function $V \to T$ that makes only a finite number of changes, i.e., σ has the property that $dom(\sigma) = \{v \in V \mid \sigma(v) \neq v\}$ is finite. It is common to write $\sigma(v)$ as $v\sigma$. See Doets [7] for a lucid introduction to the subject of substitutions in the context of logic programming. We will use $rng(\sigma)$ for $\{v\sigma \mid v \in dom(\sigma)\}$. An explicit form (or: a representation) for substitution σ is a sequence

$$[v_1/v_1\sigma, \dots, v_n/v_n\sigma],$$

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

If $t \in T$ and σ is a substitution, then we use $t\sigma$ for the result of replacing t by its σ value in case t is a variable, and leaving t unchanged otherwise. This instruction in fact lifts σ to a function in $T \to T$. Similarly, if $\bar{t}$ is a sequence of terms, $\bar{t}\sigma$ is the sequence that results from replacing all variables in $\bar{t}$ by their σ values.

We will use [] for the substitution that changes nothing, i.e, [] is the only substitution σ with $dom(\sigma) = \emptyset$. We use σ, ρ, θ as metavariables ranging over substitutions.

Definition 1 (Substitution Representations)

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

We will write $\sigma - v_i/t_i$ for

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

The composition $\sigma \circ \theta$ of two substitutions σ and θ has its usual meaning of 'first σ, next θ', which we get by means of $v(\sigma \circ \theta) := (v\sigma)\theta$. In other words:

$$v(\sigma \circ \theta) = \begin{cases} (v\sigma)\theta & \text{if } v\sigma \in V \\ v\sigma & \text{if } v\sigma \notin V \end{cases}$$

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

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

$$[v_1/t_1\theta, \ldots, v_n/t_n\theta, w_1/r_1, \ldots, w_m/r_m]$$

the bindings $v_i/t_i\theta$ *for which* $v_i = t_i\theta$, *and the bindings* w_j/r_j *for which* $w_j \in \{v_1, \ldots, v_n\}$.

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

We are now in a position to define the language $\mathcal{L}$ of DFOL. We distinguish between DFOL atoms, DFOL blocks, DFOL choices and DFOL formulas (or sequences), and we borrow some convenient syntactic conventions from C [15], in particular the use of ; as a terminator.

Definition 3 *The language* $\mathcal{L}$ *of DFOL (given a signature of predicate names and individual constants):*

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

Since every atom has suffix ;, we can use simple concatenation for sequencing (that's the C trick).

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 $\neg\{t_1 \doteq t_2;\}$.

We write $\exists\bar{v}$; for $\exists v_1; \cdots ; \exists v_n$;, where $\bar{v} = v_1, \ldots, v_n$, and $\exists W$; for $\exists\bar{w}$;, where W finite, and $\bar{w}$ some enumeration of W.

Similarly, $\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\phi\neg\{\psi\}\}$, $\forall v\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\exists v;\phi\neg\{\psi\}\}$, $\forall\bar{v}\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\exists v;\phi\neg\{\psi\}\}$, and $\forall W\{\phi \Rightarrow \psi\}$ abbreviates $\neg\{\exists W;\phi\neg\{\psi\}\}$.

We will use U (unit) to generalize over A, B, C (atoms, blocks and choices). We can think of formula ϕ as built up from units by concatenation. To let the distinction between the four categories of expressions not hurt formula induction arguments, we will often read a unit U as the formula $U[];$, 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. Similarly, we will prefix $[];$ to DFOL formulas when the need arises, so that we can conveniently consider the form of every DFOL formula to be $\sigma;\phi$, taking $[]$ for σ. This way, we can have our cake (nice syntax) and eat it (or rather, process it, by nice induction and recursion on syntax).

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

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

Definition 4 (Semantics of DFOL)

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

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:

Definition 5 (Entailment in DFOL) *ϕ dynamically entails ψ, notation $\phi \models \psi$, :⇔*

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

2 Subtleties of 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 it looks like 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

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

yields:

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

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

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 $[x/a]$ through the example formula yields two different substitution results at the end points: $[x/a]$ indicates that substitution $[x/a]$ 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.

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

Definition 6 (Left-to-Right Snowball Substitution for DFOL)

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

The reason for calling this *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:

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

As a second example, here is how the tableau rules below deal with $\exists x; Px; \exists x; \neg Px; \phi$ on the left-hand side:

$$
\begin{array}{rcl}
\exists x; Px; \exists x; \neg Px; \phi & \leadsto & [x/d_1]; Px; \exists x; \neg Px; \phi \\
& = & Pd_1; \exists x; \neg Px; \phi \\
& \leadsto & Pd_1 \text{ in left store and process } \exists x; \neg Px; \phi \\
& \leadsto & Pd_1 \text{ in left store and process } [x/d_2]; \neg Px; \phi \\
& = & Pd_1 \text{ in left store and process } \neg Pd_2; [x/d_2]; \phi \\
& \leadsto & Pd_1 \text{ in left store, } Pd_2 \text{ in right store and process } [x/d_2]; \phi \ldots
\end{array}
$$

This syntactic substitution definition for DFOL fleshes out what has been called the 'folklore idea in dynamic logic' (Van Benthem [5]) that syntactic substitution $[v/t]$ works semantically as the program instruction $v := t$ (Goldblatt [12]), with semantics given by ${}_s[\![v := t]\!]^{\mathcal{M}}u$ iff $u = s(v/[\![t]\!]^M_s)$. To see the connection, note that $v := t$ can be viewed as DFOL shorthand for $\exists v; v = t$. To generalize this to simultaneous substitution, auxiliary variables must be used, but as we have simultaneous substitution represented in the language, we need not bother about these.

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

Lemma 7 (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 ϕ, all substitutions σ:*

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

Proof. Induction on the structure of ϕ. □

Immediately from this we get the following:

Proposition 8 *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 [18]. Here we confine ourselves to the definition of $input(\phi)$, the set of variables that have an input constraining occurrence in ϕ (with $\phi \in \mathcal{L}$), and $output(\phi)$, the set of variables that have an output constraining occurrence in ϕ (again, with $\phi \in \mathcal{L}$). Let $var(\bar{t})$ be the variables among $\bar{t}$.

Definition 9 (Input constrained variables of $\mathcal{L}$ formulas)

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

The definition of the output constrained variables of ϕ 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 invisible *cons* operator that we used to build formulas in Definition 3. See, e.g., Bird and De Moor [6]. Because the definition of formulas used just concatenation, by the associativity of concatenation we have that the *snoc* counterpart to $\mathcal{L}$ looks 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 10 (Output constrained variables of $\mathcal{L}$ formulas)

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

Now, the dynamic closure of a formula ϕ 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 ϕ 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 11 *For all $\mathcal{L}$ models $\mathcal{M}$, all valuations s, s', u, u' for $\mathcal{M}$, all $\mathcal{L}$ formulas ϕ:*

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

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

The right-to-left recursion on formulas from Definition 10 suggests a counterpart to the 'left to right' substitution operation we defined above. Representations for 'right to left' substitutions are given by:

Definition 12 (Right-to-Left Substitution Representations)

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

The interpretation of $\sigma^{\check{}}$ is given by:

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

Note that the right-to-left substitution $[t \backslash v]$ would correspond to the DFOL formula $v \doteq t; \exists v;$.

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 6 above.

With right-to-left induction on $\mathcal{L}$ structure, we can now prove a right-to-left counterpart to Lemma (7):

Lemma 13 (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 ϕ, all right-to-left substitutions $\sigma^{\check{}}$:*

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

For more on this topic, see Visser [18] and [19], 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.

3 Modelling 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 [3], or Hankin [14] for a concise account), for a lambda redex $(\lambda x.t)u$ suspends a substitution $[x/u]$, 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 suspended 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 14 (Notions of Reduction for $\mathcal{L}$)

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

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

Definition 15 (Single Step R Reduction)

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

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

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

Definition 16 (R Reduction)

$$\frac{\phi_1 \xrightarrow{R} \phi_2}{\phi_1 \overset{R}{\twoheadrightarrow} \phi_2} \qquad \frac{}{\phi \overset{R}{\twoheadrightarrow} \phi} \qquad \frac{\phi_1 \overset{R}{\twoheadrightarrow} \phi_2 \quad \phi_2 \overset{R}{\twoheadrightarrow} \phi_3}{\phi_1 \overset{R}{\twoheadrightarrow} \phi_3}$$

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

Definition 17 (R Convertibility)

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

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 18 (Computation Lemma for DFOL)

Suppose $\phi \overset{R}{\twoheadleftarrow\!\!\twoheadrightarrow} \psi$.

Then for all models $\mathcal{M}$, all variable states s, u for $\mathcal{M}$, ${}_s[\![\phi]\!]_u^{\mathcal{M}}$ iff ${}_s[\![\psi]\!]_u^{\mathcal{M}}$.

Proof. Induction on the length of the computation path that connects ϕ_1 to ϕ_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 [2]. The following results will not come as a surprise.

Lemma 19 *$\xrightarrow{R}$ 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:

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

□

Lemma 20 *$\xrightarrow{R}$ is locally confluent.*

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

Theorem 21 $\xrightarrow{R}$ *is confluent.*

Proof. From Lemma 19 and Lemma 20 by Newman's lemma, which states that any terminating locally confluent relation is confluent. (See, e.g., Baader and Nipkov [2], Chapter 2.) □

From Theorem 21 it follows that $\mathcal{L}$ formulas ϕ 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:

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

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 22 (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.

4 Adaptation of Tableau Reasoning to a Dynamic Setting

In classical tableau theorem proving, when investigating whether ϕ logically implies ψ, one systematically explores possibilities to make ϕ true and ψ false. If all such explorations fail, we conclude that ψ does indeed follow from ϕ, 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 [4]).

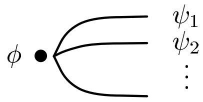
In the course of dealing with the original ϕ and ψ we decompose them into parts, so in general the data structure we deal with in classical tableau proving has the form $\Phi \bullet \Psi$, where Φ, Ψ are finite sets of formulas, with Φ the formulas we are committed to make true and Ψ the formulas we are committed to make false. The tableau rule for disjunction in classical logic illustrates this:

$$\vee_L \quad \begin{array}{c} \Phi, \alpha \vee \beta \bullet \Psi \\ \Phi, \alpha \bullet \Psi \qquad \Phi, \beta \bullet \Psi \end{array} \qquad\qquad \vee_R \quad \begin{array}{c} \Phi \bullet \alpha \vee \beta, \Psi \\ \Phi \bullet \alpha, \beta, \Psi \end{array}$$

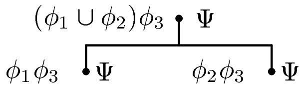
In the dynamic version of FOL, order matters. Suppose Φ would consist of $\exists x; Px;$ and $\neg Px;$, and Ψ of $Qx; \cup Sx;$. Then it matters whether we read Φ with $\exists x; Px;$ first or with $\neg Px;$ first. In the first case, we get the formula $\exists x; Px; \neg Px;$, which should lead to a contradiction, in the second case we get $\neg Px; \exists x; P; x$, which has a model that contains both Ps and non-Ps. Supposing we take the non-contradictory reading

$\neg Px; \exists x; Px;$ for the left-hand side and apply the $\cup$ analogue of $\vee$ to $\neg Px; \exists x; Px; \bullet Qx; \cup Sx;$. Then the result $\neg Px; \exists x; Px; \bullet Qx;, Sx;$ should be read in such a way that both $Qx;$ and $Sx;$ are in the dynamic scope of the quantifier $\exists x;$ that binds across the $\bullet$.

Thus, we see that the appropriate data-structure for dynamic tableau reasoning is $\phi \bullet \Psi$, where dynamic quantifiers in ϕ can bind in any of the 'trails' in Ψ. Think of it like this:



Format of Tableau Rules For convenience of presentation, we will assume that tableau trees grow upward, like apple trees. Thus, we will present a tableau splitting rule like



as:

$$\frac{\phi_1\phi_3 \bullet \Psi \qquad\qquad \phi_2\phi_3 \bullet \Psi}{(\phi_1 \cup \phi_2)\phi_3 \bullet \Psi}$$

Storage of Suspended Substitutions Since performing substitutions may be expensive, it will be advantageous to only perform a substitution σ on ϕ when needed. Rather than storing $!\sigma\phi$, the tableau rules will store $\sigma; \phi$ as an $\mathcal{L}$ formula to be processed along the lines of the principles from Section 3.

This works well for the left-hand side ϕ, but recall that the meanings of the right-hand side 'trails' ψ_i depend on the substitution that results from processing the left-hand side formula ϕ. As long as this substitution has not been fully computed, we must leave a *hole* for it, to be filled later. We do this by inserting a *substitution dummy* \$ at the place of the substitution. The presence of this dummy will allow us to move negated formulas from right to left, while remembering that they should be interpreted under a substitution representing the bindings across the $\bullet$.

So, our inquiry whether ϕ dynamically implies ψ starts on the left-hand side with a formula $[]; \phi$, and on the righthand side with a formula set $\{\$; \psi\}$. On the left we start with the empty suspended substitution, and let this substitution grow as we go along. On the right, we start with the dummy substitution $\{v \mapsto v\$ \mid v \in V\}$, to be replaced later by the substitution that results when ϕ has been fully decomposed.

Until the point where \$ gets replaced by a computed answer substitution, we will treat expressions of the form $v\$$ as variables. Thus, $\mathit{var}(\bar{t})$ may yield variables with a \$ suffix.

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 ϕ dynamically implies ψ will build a domain with positive and negative facts. As we will see, our DFOL tableaus to confute $\phi \models \psi$ will have the property that the members of $\mathit{output}(\phi) \cup \mathit{input}\psi$ (the ϕ variables constraining the output of ϕ or the input of ψ) will end up as members of the domain.

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_{new} as a generic renaming of variable v. We use C^v_{new} for the result of replacing v everywhere in C by v_{new}.

The current domain always consists of all the names (variables, possibly under a renaming, and contants) occurring in the current database C of literals (Predicate and equality atoms and their negations). We will use $d \in D$ for: 'd is a name occurring in a literal in C'.

Structure of Tableau Nodes A DFOL tableau node is a triple C, ϕ, Ψ, with C a set of literals, ϕ the antecedent (left-hand side) formula, and Ψ the set of consequent (right-hand side) formulas.

Without loss of generality, we can assume that ϕ is of the form $\sigma; \phi'$, and similarly for the members of ψ (prefix with the empty substitution if the need arises).

For a convenient distinction of left and right sides of tableau nodes, we write a triple C, ϕ, Ψ as:

$$C \mid\mid \phi \bullet \Psi.$$

Intuitively, C represents the database with information about the current node, ϕ represents the formula that is assumed to be *true* at the node, and Ψ is the set of all continuations of ϕ that are supposed to be *false* at the node.

Because our databases C of literals contain variables and identities, we need some preliminary definitions in order to define closure of a database.

Definition 23 *Let C be a database. Substitution σ is suspended in C if for all $v \in dom(\sigma)$ it holds that there is a path $t_1, \dots, t_n$ with $t_1 = v$, $t_n = v\sigma$, and either $t_i \doteq t_{i+1} \in C$ or $t_{i+1} \doteq t_i \in C$ for $1 \leq i < n$.*

Definition 24 *A database C is closed if for some σ suspended in C there is a pair of term sequences $\bar{t}, \bar{t}'$ and a predicate letter P with $\bar{t}\sigma = \bar{t}'\sigma$, $P\bar{t} \in C$, $\neg P\bar{t}' \in C$, or a pair of terms t_1, t_2 with $t_1\sigma = t_2\sigma$ and $t_1 \neq t_2 \in C$.*

Definition 25 *A tableau node is* closed *if its 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 $\exists v; \psi$ on the right-hand side, i.e., our initial node is of the form $[]; []; \bullet\{\$; \exists v; \psi\}$. In such a case it is allowed to assume the presence of a pre-given element d_0. 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.

In general, we start the tableau search procedure with a triple of the form

$$\emptyset \mid\mid []; \phi \bullet \{\$; \psi\}.$$

The initial setting with the left-hand side formula of the form $[]; \phi$ indicates that we start out the computation with the empty suspended substitution.

We use $C + P\bar{t};$ for $C \cup \{P\bar{t};\}$, and similarly for $\Psi + \psi$.

5 Tableau Rules for DFOL

We will see that substitutions get pushed from left to right in the tableaus, according to the principles explained in Section 3, and can be viewed as computed results at the open nodes. Keys to the calculus are the theory of suspended substitution developed in Section 3, and the use of a dummy substitution $v \mapsto v\$$ as a proxy for the binding substitution across the $\bullet$ that is still to be computed.

Closure

$$\frac{}{C \mid\mid \phi \bullet \Psi} \; C \text{ closed}$$

Soundness: obvious from the fact that closure of a database C represents inconsistency of the data.

Shift Substitution Across $\bullet$ The presence of $(\sigma;\rho;)$ (with special case $\sigma;[];$, substitution at the empty formula list) on the left-hand side signifies that decomposition of the left-hand dynamic formula has come to an end, and that the computation of the left-hand side substitution $\sigma \circ \rho$ has been completed. At this stage we can shift the substitution to the right-hand side. This is done by applying the map $v\$ \mapsto v\sigma\rho$. Note that the dummy variables not only are present in the righthand side Ψ of the tableau node, but may also have ended up in the facts in the database C. Thus, the $v\$ \mapsto v\sigma\rho$ map will have to be applied on Ψ and C. We use $C^{\$}_{\sigma\rho}$, $\Psi^{\$}_{\sigma\rho}$, for the results of these applications. Note that the instantiation of C to $C^{\$}_{\sigma\rho}$ may result in closure of the database.

$$\frac{C^{\$}_{\sigma\rho} \mid\mid \sigma \circ \rho; \bullet \Psi^{\$}_{\sigma\rho}}{C \mid\mid \sigma; \rho; \bullet \Psi}$$

Soundness: the map $v\$ \mapsto v\sigma$ is in agreement with the intended meaning of the $v\$$ variables.

ρ Left and Right

$$\frac{C \mid\mid (\sigma \circ \rho; \phi) \bullet \Psi}{C \mid\mid (\sigma; \rho; \phi) \bullet \Psi} \qquad \frac{C \mid\mid \phi \bullet \Psi + (\sigma \circ \rho; \psi)}{C \mid\mid \phi \bullet \Psi + (\sigma; \rho; \psi)}$$

Soundness of left-hand rule and righthand rule: substitutions always succeed, and the result of first performing substitution σ and next substitution ρ is the same as that of performing the singls substitution $\sigma \circ \rho$.

Predicate Left and Right Predicates left are shifted to the database. Predicates right give rise to branching.

$$\frac{C + P\bar{t}\sigma; \mid\mid \sigma; \phi \bullet \Psi}{C \mid\mid \sigma; P\bar{t}; \phi \bullet \Psi} \qquad \frac{C + \neg P\bar{t}\sigma; \mid\mid \phi \bullet \Psi \quad C \mid\mid \phi \bullet \Psi + \sigma; \psi}{C \mid\mid \phi \bullet \Psi + \sigma; P\bar{t}; \psi}$$

Soundness: obvious.

Equality Left and Right

$$\frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \sigma \circ [v/t_2\sigma]; \phi \bullet \Psi}{C \mid\mid \sigma; t_1 \doteq t_2; \phi \bullet \Psi}\; t_1\sigma = v \qquad \frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \sigma \circ [v/t_1\sigma]; \phi \bullet \Psi}{C \mid\mid \sigma; t_1 \doteq t_2; \phi \bullet \Psi}\; t_2\sigma = v$$

$$\frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \sigma; \phi \bullet \Psi}{C \mid\mid \sigma; t_1 \doteq t_2; \phi \bullet \Psi}\; t_1\sigma \text{ not a variable, } t_2\sigma \text{ not a variable,}$$

Soundness: adding $t_1\sigma \doteq t_2\sigma$ to the database represents storage of the meaning of $t_1 \doteq t_2$ under σ. The change of σ to $\sigma \circ [v/t_i\sigma]$ reflects the fact that $t_1\sigma \doteq t_2\sigma$ gives us the information that we need to instantiate v. Note that in cases of equalities $v \doteq w$ with $v \notin \mathit{dom}(\sigma)$ and $w \notin \mathit{dom}(\sigma)$ it does not matter whether we apply the rule that inserts substitution $[v/w]$ or the one that inserts $[w/v]$.

$$\frac{C + t_1\sigma \neq t_2\sigma; ,\ \mid\mid \phi \bullet \Psi \qquad C \mid\mid \phi \bullet \Psi + \sigma; \psi}{C \mid\mid \phi \bullet \Psi + \sigma; t_1 \doteq t_2; \psi}$$

Soundness: obvious.

Quantifiers Left and Right

$$\frac{C^v_{\text{new}} \mid\mid \sigma - v/v\sigma; \phi \bullet \Psi^v_{\text{new}}}{C \mid\mid \sigma; \exists v; \phi \bullet \Psi}$$

Here C^v_{new} denotes the result of replacing v everywhere in C by v_{new}, and Ψ^v_{new} represents the set

$$\{[v/v_{\text{new}}] \circ \sigma; \psi \mid \sigma; \psi \in \Psi, \sigma \neq \$\} \cup \{\sigma; \psi \mid \sigma; \psi \in \Psi, \sigma = \$\}.$$

Soundness: the effect of quantifier $\exists v$ is that it cuts the dynamic link between previous occurrences of v and occurrences to come. This gets reflected by a renaming operation on all the previously active occurrences of v. The information about v in σ is discarded, as this information concerns the incarnation of v that ceases to be active.

$$\frac{C \mid\mid \phi \bullet \Psi + \sigma; \exists v; \psi + [v/d] \circ \sigma; \psi}{C \mid\mid \phi \bullet \Psi + \sigma; \exists v; \psi}\; d \text{ occurs in } C$$

Soundness: to make $\sigma; \exists v; \psi$ false at the current node, we have to make ψ false for any d currently in the database, and for all elements d to be added later on.

Important remark Quantification right 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.

In Section 8 we will see how in certain cases we can 'tune the engine' in such a way that the rule is only triggered for a 'relevant subset' of domain elements.

Choice Left and Right

$$\frac{C \mid\mid \sigma; \phi_1\phi_3 \bullet \Psi \qquad C \mid\mid \sigma; \phi_2\phi_3 \bullet \Psi}{C \mid\mid \sigma; (\phi_1 \cup \phi_2)\phi_3) \bullet \Psi}$$

Soundness: immediate from the computation lemma for DFOL. Note how the rule effectively splits the computation, with distribution of σ over the two trails.

$$\frac{C \mid\mid \phi \bullet \Psi + \sigma; \psi_1\psi_3 + \sigma; \psi_2\psi_3}{C \mid\mid \phi \bullet \Psi + \sigma; (\psi_1 \cup \psi_2)\psi_3}$$

Soundness: immediate from the computation lemma for DFOL.

Negation Left and Right Negations on the left are moved to the right, together with the substitution that suspends their current bindings. The { } wrapper is kept, to indicate that the computed substitution does not constrain the output. (But note that as the negated subformula is moved to the end of a trail, there is no danger of wrong dynamic bindings further on.)

$$\frac{C \mid\mid \sigma; \phi_2 \bullet \Psi + \{\sigma; \phi_1\}}{C \mid\mid \sigma; \neg\{\phi_1\}\phi_2 \bullet \Psi}$$

Soundness: suppose $\neg\{\phi_1\}\phi_2$ is true under σ. Then it follows from the computation rule for DFOL that ϕ_1 is false under σ and ϕ_2 is true under σ.

Negations on the right are moved to the left, with their suspended substitution, and with a protection of the context against their dynamic force by means of a { } wrapper. This is needed, as the environment has to be protected against dynamic binding from within the moved element.

We make a distinction between the case where the suspended substitution equals \$ and the case where it does not:

$$\frac{C \mid\mid \phi\{\psi_1\} \bullet \Psi \qquad C \mid\mid \phi \bullet \Psi + \$; \psi_2}{C \mid\mid \phi \bullet \Psi + \$; \neg\{\psi_1\}\psi_2} \qquad \frac{C \mid\mid \{\sigma; \psi_1\}\phi \bullet \Psi \qquad C \mid\mid \phi \bullet \Psi + \sigma; \psi_2}{C \mid\mid \phi \bullet \Psi + \sigma; \neg\{\psi_1\}\psi_2}\ \sigma \neq \$$$

Soundness: suppose $\neg\{\psi_1\}\psi_2$ is false under \$. Since \$ is a dummy for the binding across $\bullet$, we can shift ψ_1 across the bullet while removing the dummy substitution. Suppose $\neg\{\psi_1\}\psi_2$ is false under $\sigma \neq \$$. Then it follows from the computation rule for DFOL that either ϕ_1 is true under σ or ϕ_2 is false under σ. Note that $\{\sigma; \psi_1\}$ carries its own dynamic context, in the form of the substitution σ, and does not influence what follows. The surrounding context is protected from its dynamic force by the presence of the { } wrapper. For these reasons it is sound to put $\{\sigma; \psi_1\}$ as a prefix to the antecedent.

Block The rules for block formulas use recursion on the structure of the block. The first rule distributes a substitution over a block, the others deal with formulas starting with a block.

$\sigma\{\phi\}$ Left and Right

$$\frac{C \mid\mid \{\sigma; \phi_1\}\sigma; \phi_2 \bullet \Psi}{C \mid\mid \sigma; \{\phi_1\}\phi_2 \bullet \Psi} \qquad \frac{C \mid\mid \phi \bullet \Psi + \{\sigma; \psi_1\}\sigma; \psi_2}{C \mid\mid \phi \bullet \Psi + \sigma; \{\psi_1\}\psi_2}$$

Soundness: immediate from the computation lemma for DFOL.

$\{\sigma;\}$ **Left and Right**

$$\frac{C \mid\mid \phi \bullet \Psi}{C \mid\mid \{\sigma;\}\phi \bullet \Psi} \qquad \frac{C \mid\mid \phi \bullet \Psi + \psi}{C \mid\mid \phi \bullet \Psi + \{\sigma;\}\psi}$$

Soundness: immediate from the computation lemma for DFOL.

$\{\sigma;\rho;\phi\}$ **Left and Right**

$$\frac{C \mid\mid \{\sigma \circ \rho; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; \rho; \phi_1\}\phi_2 \bullet \Psi} \qquad \frac{C \mid\mid \phi \bullet \Psi + \{\sigma \circ \rho; \psi_1\}\psi_2}{C \mid\mid \phi \bullet \Psi + \{\sigma; \rho; \psi_1\}\psi_2}$$

Soundness: immediate from the computation lemma for DFOL.

$\{\sigma; P\bar{t}; \phi\}$ **Left and Right**

$$\frac{C + P\bar{t}\sigma; \mid\mid \{\sigma; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; P\bar{t}; \phi_1\}\phi_2 \bullet \Psi} \qquad \frac{C + \neg P\bar{t}\sigma; \mid\mid \phi \bullet \Psi \quad C \mid\mid \phi \bullet \Psi + \{\sigma; \psi_1\}\psi_2}{C \mid\mid \phi \bullet \Psi + \{\sigma; P\bar{t}; \psi_1\}\psi 2}$$

Soundness: obvious.

$\{\sigma; t_1 \doteq t_2; \phi\}$ **Left and Right**

$$\frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \{\sigma \circ [v/t_2\sigma]; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; t_1 \doteq t_2; \phi_1\}\phi_2 \bullet \Psi}\ t_1\sigma = v$$

$$\frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \{\sigma \circ [v/t_1\sigma]; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; t_1 \doteq t_2; \phi_1\}\phi_2 \bullet \Psi}\ t_2\sigma = v$$

$$\frac{C + t_1\sigma \doteq t_2\sigma \mid\mid \{\sigma; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; t_1 \doteq t_2; \phi_1\}\phi_2 \bullet \Psi}\ t_1\sigma \text{ not a variable, } t_2\sigma \text{ not a variable,}$$

$$\frac{C + t_1\sigma \neq t_2\sigma; \mid\mid \phi \bullet \Psi \qquad C \mid\mid \phi \bullet \Psi + \{\sigma; \psi_1\}\psi_2}{C \mid\mid \phi \bullet \Psi + \{\sigma; t_1 \doteq t_2; \psi_1\}\psi_2}$$

Soundness: as for equalities

$\{\sigma; \exists v; \phi\}$ **Left and Right**

$$\frac{C \mid\mid \{[v/v_{\text{new}}] \circ \sigma; \phi_1\}\phi_2 \bullet \Psi}{C \mid\mid \{\sigma; \exists v; \phi_1\}\phi_2 \bullet \Psi}$$

$$\frac{C \mid\mid \phi \bullet \Psi + \{\sigma; \exists v; \psi_1\}\psi_2 + \{[v/d] \circ \sigma; \psi_1\}\psi_2}{C \mid\mid \phi \bullet \Psi + \{\sigma; \exists v; \psi_1\}\psi_2}\ d \text{ occurs in } C$$

Note how the crucial difference between $\exists v; \phi$ and $\{\exists v; \phi\}$ shows up. In $\exists v; \phi$, the v-occurrences bound in ϕ by $\exists v$ are dynamically bound. The current dynamically bound variables are used as names for domain elements. In $\{\exists v; \phi\}$ the v-occurrences bound in ϕ by $\exists v$ are bound in exactly the same way as in classical FOL. The list of active variable occurrences remains the same, for the binding of v inside the block does not affect the situation outside.

The change in the suspended substitution under the test does not affect the suspended substitution outside. By the computation lemma for DFOL this is sound.

$\{\sigma; (\phi_1 \cup \phi_2)\phi_3\}$ **Left and Right**

$$\frac{C \mid\mid \{\sigma; \phi_1\phi_3\}\phi_4 \bullet \Psi \quad C \mid\mid \{\sigma; \phi_2\phi_3\}\phi_4 \bullet \Psi}{C \mid\mid \{\sigma; (\phi_1 \cup \phi_2)\phi_3\}\phi_4 \bullet \Psi}$$

Differs only from the rule for choice left in the fact that the suspended substitution that gets distributed remains isolated from the further context. This is sound by the computation lemma for DFOL.

$\{\sigma; \neg\{\phi_1\}\phi_2\}$ **Left and Right**

$$\frac{C \mid\mid \{\sigma; \phi_2\}\phi_3 \bullet \Psi + \{\sigma; \phi_1\}}{C \mid\mid \{\sigma; \neg\{\phi_1\}\phi_2\}\phi_3 \bullet \Psi}$$

$$\frac{C \mid\mid \phi\{\psi_1\} \bullet \Psi \qquad C \mid\mid \phi \bullet \Psi + \{\$; \psi_2\}\psi_3}{C \mid\mid \phi \bullet \Psi + \{\$; \neg\{\psi_1\}\psi_2\}\psi_3}$$

$$\frac{C \mid\mid \{\sigma; \psi_1\}\phi \bullet \Psi, R \qquad C \mid\mid \phi \bullet \Psi + \{\sigma; \psi_2\}\psi_3}{C \mid\mid \phi \bullet \Psi + \{\sigma; \neg\{\psi_1\}\psi_2\}\psi_3}\ \sigma \neq \$$$

Soundness: Negation under a test behaves the same as plain negation. This reflects the semantic fact that $\{\neg\{\}\}$, i.e., $\neg\neg\neg\{\}$, boils down to just the negation test in DFOL. Thus, the rules are sound.

$\{\sigma; \{\phi_1\}\phi_2\}$ **Left and Right**

$$\frac{C \mid\mid \{\sigma; \phi_1\}\{\sigma; \phi_2\}\phi_3 \bullet \Psi}{C \mid\mid \{\sigma; \{\phi_1\}\phi_2\}\phi_3 \bullet \Psi}$$

$$\frac{C \mid\mid \phi \bullet \Psi + \{\sigma; \psi_1\}\{\sigma; \psi_2\}\psi_3}{C \mid\mid \phi \bullet \Psi + \{\sigma; \{\psi_1\}\psi_2\}\psi_3}$$

By the computation lemma for DFOL, both of these rules are sound.

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

Theorem 26 (Soundness Theorem) *The tableau calculus for DFOL is sound:*

> *If the tableau for* $\emptyset \mid\mid []; \phi \bullet \{\$; \psi\}$ *closes then* $\phi \models \psi$.

6 Examples

Consider the Aristotelian 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 (omitted)

More examples ...

7 Completeness

*** Section under Construction ***

Completeness: we confine ourselves to an outline of the proof.

First 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 and constants Cons, then the trace sets for the language will have terms ${}^{\infty}V \cup \text{Cons}$. In other words, in the traces, we are allowed to make as many copies of the variables in V as we like.

Next prove the Trace Lemma:

Lemma 27 (Trace Lemma) *Every trace set T over a set of terms U is satisfiable in a domain $U_{\equiv}$, where $\equiv$ is the equivalence given by $t \equiv t'$ iff $t\sigma = t'\sigma$, for a substitution σ suspended in T.*

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. The crucial issue is that all infinite branches must be fair, in the sense that all $\exists$ right obligations for new individuals v_{new} must be met along the branch. A fair DFOL tableau is finished if all its infinite branches are fair, and none of its finite branches can be extended further by means of a rule application.

Theorem 28 *Any open branch in a finished fair DFOL tableau corresponds to a trace set.*

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

Theorem 29 *Any open branch in a finished fair DFOL tableau is simultaneously satisfiable.*

Theorem 30 (Completeness) *If $\phi \models \psi$ then the tableau for $\emptyset \ || \ []; \phi \bullet \{\$; \psi\}$ closes.*

8 Tuning the Engine

*** Section under Construction ***

9 Higher Order Extensions

*** Section under Construction ***

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 $P\bar{t}\sigma$ (ground equalities $t_1\sigma \doteq t_2\sigma$) on the left-hand side, we check the model for $\mathcal{M} \models P\bar{t}\sigma$ (for $[\![t_1\sigma]\!]^{\mathcal{M}} = [\![t_2\sigma]\!]$), and close the branch if the test fails, continue otherwise. Similarly, instead of storing ground predicates $P\bar{t}\sigma$ (ground equalities $t_1\sigma \doteq t_2\sigma$) on the left-hand side, we check the model for $\mathcal{M} \not\models P\bar{t}\sigma$ (for $[\![t_1\sigma]\!]^{\mathcal{M}} \neq [\![t_2\sigma]\!]$), and close the branch if the test fails, continue otherwise.

11 Connection to Dynamic Logic Programming

Dynamo is a programming language based on DFOL. Earlier implementations were described in [8, 10]. The DFOL tableau calculus presented here constitutes the theoretical basis for *Dynamo*.

The rather naive versions of Dynamo implemented so far implement tableau reasoning for DFOL with respect to a fixed model.

The queries are of the simplest possible kind: [];.

To find the answer to this query, given a formula ϕ considered as Dynamo program data, Dynamo essentially puts the tableau calculus to work on:

$$[]; \phi \bullet \{\perp\},$$

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 ϕ is inconsistent (with the information obtained from testing on the natural numbers), and Dynamo reports 'false'. If the tableau remains open, Dynamo reports that ϕ 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 constitute the computed answers.

Description of new implementation ...

12 Related Work

Comparison with the calculus of Van Eijck [9] The calculus of Van Eijck [9] 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 suspended substutitions and protect outside environments from dynamic side effecs by means of block operations. In a sense, the present calculus offers a full account of the phenomenon of local variable use in DFOL.

Comparison with Apt and Bezem [1] Our treatment of equality uses a generalization of a stratagem from Apt and Bezem [1]: in the context of a partial variable map σ, they call $v \doteq t$ a σ *assignment* if $v \notin dom(\sigma)$, and all variables occurring in t are in $dom(\sigma)$. We can generalize this because our substitutions are total.

Comparison with tableau reasoning for modal logics ...

Comparison with labeled deduction for the guarded fragment ...

Comparison of Dynamo with Alma-0 ...

13 Conclusion

...

Acknowledgments

Thanks to Johan van Benthem, Michael Kohlhase, Maarten Marx, Joachim Niehren, Kees Vermeulen, Albert Visser, ...

Proposition (8) was triggered by a question from Krzysztof Apt.

References

[1] 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`.

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

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

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

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

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

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

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

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

[10] J. van Eijck. Computing with dynamic first order logic. submitted to Proceedings Eighth CSLI Workshop on Logic, Language and Computation, 2000.

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

[12] 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.

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

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

[15] B. W. Kernighan and D. M. Ritchie. *The C programming language.* Prentice Hall, Englewood Cliffs (NJ), 1978.

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

[17] 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.

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

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