Automated Synthesis of Tableau Calculi

Renate A. Schmidt[1] and Dmitry Tishkovsky[1]

School of Computer Science, The University of Manchester

Abstract This paper presents a method for synthesising sound and complete tableau calculi. Given a specification of the formal semantics of a logic, the method generates a set of tableau inference rules which can then be used to reason within the logic. The method guarantees that the generated rules form a calculus which is sound and constructively complete. If the logic can be shown to admit finite filtration with respect to a well-defined first-order semantics then adding a general blocking mechanism produces a terminating tableau calculus. The process of generating tableau rules can be completely automated and produces, together with the blocking mechanism, an automated procedure for generating tableau decision procedures. For illustration we show the workability of the approach for propositional intuitionistic logic.

1 Introduction

We are interested in the problem of automatically generating a tableau calculus for a logic. We assume that the logic is defined by a high-level specification of the formal semantics. Our aim is to turn this into a set of inference rules that provide a sound and complete tableau calculus for the logic. For a decidable logic we want to generate a terminating calculus. In previous work we have described a framework for turning sound and complete tableau calculi into decision procedures [6]. The prerequisites for this to work are that the logic admits the effective finite model property shown by a filtration argument, and that (i) the tableau calculus is sound and constructively complete, and (ii) a weak form of subexpression property holds for tableau derivations. Constructive completeness is a slightly stronger notion than completeness and means that for every open branch in a tableau there is a model which reflects all the expressions (formulae) occurring on the branch. The subexpression property says that every expression in a derivation is a subexpression of the input expression with respect to a finite subexpression closure operator.

In order to be able to exploit the 'termination through blocking' results in [6], in this paper, our goal is to produce tableau calculi that satisfy the prerequisites (i) and (ii). It turns out that provided that the semantics of the logic is well-defined in a certain sense, the subexpression property can be imposed on the generated calculus. Crucial is the separation of the syntax of the logic from the 'extras' in the meta-language needed for the semantic specification of the logic. The process can be completely automated and gives, together with the unrestricted blocking mechanism and the results in [5,6], an automated procedure for generating tableau decision procedures for logics, whenever they have the effective finite model property with respect to a well-defined first-order semantics.

That the generated calculi are constructively complete has the added advantage that models can be effectively generated from open, finished branches in tableau derivations. This means that the synthesised tableau calculi can be used for model building purposes.

The method works as follows. The user defines the formal semantics of the given logic in a many-sorted first-order language so that certain well-definedness conditions hold. The method automatically reduces the semantic specification of the logic to Skolemised implicational forms which are then rewritten as tableau inference rules. Combined with some default closure and equality rules, this provides a sound and constructively complete calculus for the logic. Under certain conditions it is then possible to refine the rules. If the logic can be shown to admit finite filtration, then the generated calculus can be automatically turned into a terminating calculus by adding the unrestricted blocking mechanism from [5].

The method is intended to be as general as possible, and cover as many logics as possible. Our main applications are non-classical logics and description logics. As a case study we consider the application of the method to propositional intuitionistic logic (e.g. [3]). Intuitionistic logic provides a nearly perfect example because the semantics of the logical connectives is not Boolean and the semantics is restricted by a background theory. In addition, the logic is simple.

The paper is structured as follows. Section 2 defines the apparatus for specifying the semantics of the logic of interest. Section 3 is about synthesising tableau rules. In Section 4 we prove that the generated rules form a sound and constructively complete calculus for the logic. Section 5 discusses ways of refining the rules in order to reduce branching and redundancy in the syntax of the calculus. In Section 6 the approach is applied to intuitionistic logic. We conclude with a discussion of the method.

This paper contains no proofs, but these are given in the long version [7]. The long version also contains more examples.

2 Specifying the Semantics of the Logic

For the sake of generality we assume the logic for which we want to develop a tableau calculus is a many-sorted logic.

Let $\mathsf{Sorts} \stackrel{\mathsf{def}}{=} \{0, 1, \ldots, N\}$ be an index set of sorts and Conn a countable set of the logical connectives of the logic. Every connective σ in Conn is associated with a tuple $(i_1, i_2, \ldots, i_{m+1}) \in \mathsf{Sorts}^{(m+1)}$, where $m \geq 0$. The last argument i_{m+1} is the sort of the expression obtained by applying σ to expressions of sorts $i_1, i_2, \ldots, i_m$, respectively. We say that σ is an m-ary connective of sort $(i_1, i_2, \ldots, i_{m+1})$.

By $\mathcal{L}$ we denote an *abstract sorted language* defined over an alphabet given by a set of sorts Sorts, a set of connectives Conn, a countable set of variable symbols $\{p^i_j \mid i \in \mathit{Sorts}, j \in \omega\}$, and a countable set of constant symbols $\{q^i_j \mid i \in \mathit{Sorts}, j \in \omega\}$. $\mathcal{L}$ is defined as a set of *expressions* over the alphabet closed under the connectives in Conn. More formally, let $\mathcal{L} \stackrel{\text{def}}{=} \bigcup_{i \in \mathit{Sorts}} \mathcal{L}^i$, where each $\mathcal{L}^i$ denotes a *set of expressions of sort* i defined as the smallest set of expressions satisfying the following conditions:

- All variables p^i_j and all constants q^i_j in the alphabet belong to $\mathcal{L}^i$.
- For every connective $\sigma \in \mathit{Conn}$ of sort $(i_1, i_2 \ldots, i_{m+1})$, $\sigma(E_1, \ldots, E_m)$ belongs to $\mathcal{L}^{i_{m+1}}$, whenever $E_1, \ldots, E_m$ belong to $\mathcal{L}^{i_1}, \ldots, \mathcal{L}^{i_m}$, respectively.

Symbols, expressions and connectives in $\mathcal{L}$ are also referred to as $\mathcal{L}$-symbols, $\mathcal{L}$-expressions and $\mathcal{L}$-connectives. Variables and constants in $\mathcal{L}$ are called *atomic* $\mathcal{L}$-expressions. We usually refer to expressions in $\mathcal{L}^0$ as *individuals*, expressions in $\mathcal{L}^1$ as *concepts*, and expressions in $\mathcal{L}^2$ as *roles*.

For an $\mathcal{L}$-expression E, the notation $E(p_1, \ldots, p_m)$ indicates that $p_1, \ldots, p_m$ are variables in the expression E. $E(E_1, \ldots, E_m)$ denotes the expression obtained by uniformly substituting E_i into p_i, for all $i = 1, \ldots, m$. Similarly, if X is a set of $\mathcal{L}$-expressions depending on variables $p_1, \ldots, p_m$, we indicate this as $X(p_1, \ldots, p_m)$ and denote by $X(E_1, \ldots, E_m)$ the set of expressions which are instances of expressions from X under uniform substitution of expressions $E_1, \ldots, E_m$ into $p_1, \ldots, p_m$, respectively.

Let $\prec$ be any transitive ordering on $\mathcal{L}$-expressions, E an $\mathcal{L}$-expression, and X a set of $\mathcal{L}$-expressions. We define $\mathsf{sub}_{\prec}(E) \stackrel{\text{def}}{=} \{E' \mid E' \prec E\}$ and $\mathsf{sub}_{\prec}(X) \stackrel{\text{def}}{=} \bigcup_{E \in X} \mathsf{sub}_{\prec}(E)$. We write $\mathsf{sub}_{\prec}(E_1, \ldots, E_m)$ rather than $\mathsf{sub}_{\prec}(\{E_1, \ldots, E_m\})$.

The language in which the semantics of the given logic is specified, is a sorted first-order language with equality, denoted by $\mathit{FO}(\mathcal{L})$. Formally, $\mathit{FO}(\mathcal{L})$ is an extension of the language $\mathcal{L}$ with: one additional sort, additional symbols, the usual connectives and quantifiers of first-order logic, and the equality predicate. The sorts of $\mathit{FO}(\mathcal{L})$ are $\mathit{Sorts} \cup \{N+1\} = \{0, \ldots, N, N+1\}$. We call the additional sort $N+1$ the *designated sort*, and symbols that operate on this sort, *designated symbols*. The additional symbols comprise of a countable set of variable symbols $\{x, y, z, x_0, y_0, z_0, \ldots\}$ of the designated sort, a countable set of constants $\{a, b, c, a_0, b_0, c_0, \ldots\}$ of the designated sort, function symbols $\{f, g, h, f_0, g_0, h_0, \ldots\}$ mapping argument terms to terms of sort $N+1$, and a countable set of constant predicate symbols $\{P, Q, R, P_0, Q_0, R_0, \ldots\}$ of the designated sort (i.e., argument terms are required to be terms of sort $N+1$). In addition, $\mathit{FO}(\mathcal{L})$ contains intersort symbols denoted by $\nu_0, \ldots, \nu_N$, i.e., one for each sort in Sorts. The purpose of the intersort symbols is to define the semantics of the connectives of the logic (similar to satisfaction conditions in standard definitions of the semantics of connectives). In particular, ν_0 is a unary function symbol of sort $(0, N+1)$, and each of the remaining ν_i is a predicate symbol of sort $(i, N+1, \ldots, N+1)$, with arity $i+1$. Furthermore, for every sort we assume the presence of a binary predicate symbol functioning as equality predicate for that sort. For reasons of simplicity, we use one symbol, namely $\approx$, for each of the equality predicates.

We fix some more common notation. $\overline{w}$ denotes a sequence of first-order variables: $\overline{w} \stackrel{\text{def}}{=} w_1, \ldots, w_n$. Similarly, $\forall \overline{w}$ denotes the universal quantifier prefix

$\forall \overline{w} \overset{\mathsf{def}}{=} \forall w_1 \cdots \forall w_n$. For any set S of formulae, $\forall S$ denotes the universal closure of S, i.e., the set $\forall S \overset{\mathsf{def}}{=} \{\forall \overline{w}\ \phi(\overline{w}) \mid \phi(\overline{w}) \in S\}$. The symbol $\sim$ denotes complementation, i.e., $\sim\psi$ denotes ψ' if $\psi = \neg\psi'$, and $\neg\psi$, otherwise.

Formulae of $\mathit{FO}(\mathcal{L})$ in which all occurrences of the $\mathcal{L}$-variables p^i_j (of sorts $i = 0, \ldots, N$) are free are called *$\mathcal{L}$-open* formulae. Similarly, any $\mathcal{L}$-open formula is an *$\mathcal{L}$-open sentence* if it does *not* have free occurrences of variables of the designated sort $N + 1$.

For any set S of $\mathcal{L}$-open formulae in $\mathit{FO}(\mathcal{L})$ and a set X of $\mathcal{L}$-expressions, let $S{\restriction}X$ be the set of substitution instances of formulae in S under substitutions into the variables of $\mathcal{L}$ which do not contain expressions outside X. Formally,

$$\begin{aligned} S{\restriction}X \ \overset{\mathsf{def}}{=}\ \{&\phi(E_1, \ldots, E_m) \mid \phi(p_1, \ldots, p_m) \in S \text{ and} \\ &\text{all } \mathcal{L}\text{-expressions occurring in } \phi(E_1, \ldots, E_m) \text{ belong to } X\}. \end{aligned}$$

The semantics of $\mathcal{L}$ is specified in $\mathit{FO}(\mathcal{L})$ as follows. Each expression in $\mathcal{L}$ is interpreted as a term in $\mathit{FO}(\mathcal{L})$. In particular, each variable symbol p^i_j in $\mathcal{L}^i$ is interpreted as a variable of sort i in $\mathit{FO}(\mathcal{L})$, each constant symbol q^i_j in $\mathcal{L}^i$ is interpreted as a constant of sort i in $\mathit{FO}(\mathcal{L})$, and every connective σ is interpreted as a function of the same sort as σ.

An *$\mathcal{L}$-structure* is a tuple $\mathcal{I} \overset{\mathsf{def}}{=} (\mathcal{L}^0, \ldots, \mathcal{L}^N, \Delta^{\mathcal{I}}, \nu_0^{\mathcal{I}}, \ldots, \nu_N^{\mathcal{I}}, a^{\mathcal{I}}, \ldots, P^{\mathcal{I}}, \ldots)$ where $\Delta^{\mathcal{I}}$ is a non-empty set, $\nu_0(\ell)^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for every individual $\ell \in \mathcal{L}^0$, $\nu_n^{\mathcal{I}} \subseteq \mathcal{L}^n \times (\Delta^{\mathcal{I}})^n$, for $0 < n \leq N$. $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $P^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^m$, where m is the arity of P. Observe that an $\mathcal{L}$-structure $\mathcal{I}$ is a first-order model (interpretation) of the language $\mathit{FO}(\mathcal{L})$. For simplicity we omit the sets $\mathcal{L}^0, \ldots, \mathcal{L}^N$ and simply write $\mathcal{I} = (\Delta^{\mathcal{I}}, \nu_0^{\mathcal{I}}, \ldots, \nu_N^{\mathcal{I}}, a^{\mathcal{I}}, \ldots, P^{\mathcal{I}}, \ldots)$.

A *valuation* in $\mathcal{I}$ is a mapping ι from the set of variables and constants of $\mathit{FO}(\mathcal{L})$ to $\mathcal{L} \cup \Delta^{\mathcal{I}}$ such that $\iota(p^i_j), \iota(q^i_j) \in \mathcal{L}^i$, and $\iota(x_j), \iota(a_j) \in \Delta^{\mathcal{I}}$. We say that a valuation ι in an $\mathcal{L}$-structure is *canonical* if every variable and constant of any sort $i = 0, \ldots, N$ is interpreted by itself, that is, $\iota(p^i_j) = p^i_j$ and $\iota(q^i_j) = q^i_j$ for every variable p^i_j and constant q^i_j of the language $\mathcal{L}$. This means that a canonical valuation of every term of any sort $i = 0, \ldots, N$ is the term itself. It is not difficult to see that any $\mathcal{L}$-open formula ϕ is satisfiable in an $\mathcal{L}$-structure iff it is satisfiable in an $\mathcal{L}$-structure under a canonical valuation. We write $S \models_c S'$ for sets of formulae S and S', if, for every $\mathcal{L}$-structure $\mathcal{I}$ and a canonical valuation ι in $\mathcal{I}$, $\mathcal{I}, \iota \models S$ implies $\mathcal{I}, \iota \models S'$. Similarly, we write $\mathcal{I} \models_c S$ iff there is a canonical valuation ι such that $\mathcal{I}, \iota \models S$.

We say that a concept C is *satisfiable* in $\mathcal{I}$ if there is an $a \in \Delta^{\mathcal{I}}$ such that $(C, a) \in \nu_1^{\mathcal{I}}$, or equivalently $\mathcal{I} \models_c \exists x\ \nu_1(C, x)$. A concept C is *valid* in $\mathcal{I}$ if $\mathcal{I} \models_c \forall x\ \nu_1(C, x)$.

Let S be a set of $\mathcal{L}$-open sentences in $\mathit{FO}(\mathcal{L})$. A formula ϕ^σ in the language of S *defines the connective* σ with respect to S if it does not contain σ and the following holds:

$$(*) \qquad \forall S \models \forall p_1 \ldots \forall p_m\ \forall \overline{x}\ (\nu_n(\sigma(p_1, \ldots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \ldots, p_m, \overline{x})).$$

Here $p_1, \ldots, p_m$ are variables of appropriate sorts which match the signature of σ, and n is the result sort of σ (for $\overline{x} = (x_1, \ldots, x_n)$). We also say S *defines* σ.

$$\forall x\ (x \approx x) \qquad \forall x \forall y\ (x \approx y \rightarrow y \approx x) \qquad \forall x \forall y \forall z\ (x \approx y \wedge y \approx z \rightarrow x \approx z)$$
$$\forall x_1 \cdots \forall x_n \forall y_i\ \ (P(x_1, \dots, x_n) \wedge x_i \approx y_i \rightarrow P(x_1, \dots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p\, \forall x_1 \cdots \forall x_n \forall y_i\ \ (\nu_n(p, x_1, \dots, x_n) \wedge x_i \approx y_i \rightarrow \nu_n(p, x_1, \dots x_{i-1}, y_i, x_{i+1}, x_n))$$
$$\forall p_1 \cdots \forall p_m \forall x_1 \cdots \forall x_n \forall y_i\ (x_i \approx y_i \rightarrow$$
$$f(p_1, \dots, p_m, x_1, \dots, x_n) \approx f(p_1, \dots, p_m, x_1, \dots x_{i-1}, y_i, x_{i+1}, \dots, x_n))$$

Figure 1. Equality axioms in $\mathit{FO}(\mathcal{L})$.

The $\mathcal{L}$-open sentence $\forall \overline{x}\ (\nu_n(\sigma(p_1, \dots, p_m), \overline{x}) \equiv \phi^\sigma(p_1, \dots, p_m, \overline{x})$ is said to be a *σ-definition (in S)*.

By definition, a *(first-order) semantic specification* of $\mathcal{L}$ is a set of $\mathcal{L}$-open sentences in $\mathit{FO}(\mathcal{L})$ that defines the connectives of $\mathcal{L}$. Given a semantic specification S, we use the notation S^0 for the set of $\mathcal{L}$-open sentences defining the connectives of $\mathcal{L}$.

For the sake of generality, we always include the usual equality axioms, listed in Figure 1, in a semantic specification. This ensures that $\approx$ is a congruence on every sort in any first-order interpretation of $\mathit{FO}(\mathcal{L})$.

We consider a semantic specification S to be *normalised* if it consists of three disjoint parts. More specifically, $S = S^+ \cup S^- \cup S^b$, where S^+, S^-, and S^b are disjoint sets of sentences satisfying the following:

(n1) S^+ is a set of $\mathcal{L}$-open sentences ξ_+^E of the form:

$$\xi_+^E \overset{\mathsf{def}}{=} \forall \overline{x}\ (\nu_n(E(p_1, \dots, p_m), \overline{x}) \rightarrow \phi_+^E(p_1, \dots, p_m, \overline{x})),$$

(n2) S^- is a set of $\mathcal{L}$-open sentences ξ_-^E of the form:

$$\xi_-^E \overset{\mathsf{def}}{=} \forall \overline{x}\ (\phi_-^E(p_1, \dots, p_m, \overline{x}) \rightarrow \nu_n(E(p_1, \dots, p_m), \overline{x})),$$

(n3) None of the $\mathcal{L}$-open sentences in S^b contain non-atomic $\mathcal{L}$-expressions.

In this definition, we suppose that multiple sentences of the form (n1) (resp. (n2)) for the same expression E in S^+ and S^- are all equivalently reduced to a single sentence ξ_+^E (resp. ξ_-^E). The intuition is that S^+ and S^- define the semantics of the connectives. S^+ defines it for positive occurrences of expressions E (with free variables $p_1, \dots, p_m$), while S^- defines it for negative occurrences of expressions E. We refer to S^b as the *background theory* of the semantics S. Note that S^b includes the equality axioms.

It can be seen that the set $S^0 \cup S^b$ is a semantic specification which can be turned into normalised form by decomposing each connective definition in S^0 into two implications. In fact, S^0 and $S^+ \cup S^-$ play the same role in axiomatising $\mathcal{L}$-connectives in $\mathit{FO}(\mathcal{L})$ modulo the background theory S^b.

For every $\mathcal{L}$-expression E, let

$$\Phi_+^E \overset{\mathsf{def}}{=} \{\phi_+^F(E_1, \dots, E_m, \overline{x}) \mid E = F(E_1, \dots, E_m) \text{ for some } \xi_+^{F(p_1,\dots,p_m)} \text{ from } S\},$$
$$\Phi_-^E \overset{\mathsf{def}}{=} \{\phi_-^F(E_1, \dots, E_m, \overline{x}) \mid E = F(E_1, \dots, E_m) \text{ for some } \xi_-^{F(p_1,\dots,p_m)} \text{ from } S\}.$$

Thus, Φ_{+}^{E} (resp. Φ_{-}^{E}) is the set of instantiations of succedents (resp. antecedents) of positive (resp. negative) specifications in S, where the antecedents (resp. succedents) are unifiable with the given expression E.

The expression specifications in any normalised semantics S induce an ordering $\prec$ on expressions as follows. Let $\prec$ be the smallest transitive ordering satisfying: $E' \prec E$ whenever there is a sentence $\xi_{+}^{F(p_1,\ldots,p_m)}$ or a sentence $\xi_{-}^{F(p_1,\ldots,p_m)}$ such that $E = F(E_1, \ldots, E_m)$, for some $\mathcal{L}$-expressions $E_1, \ldots, E_m$, and E' occurs in $\phi_{+}^{F}(E_1, \ldots, E_m, \overline{x})$ or $\phi_{-}^{F}(E_1, \ldots, E_m, \overline{x})$, respectively. Because we can assume that S^0 is also a normalised semantic specification, it similarly induces an ordering $\prec_0$ which is assumed to be well-founded.

Usually the semantics is defined by induction in terms of definitions of the semantics of the connectives and the primitives in the logic which is lifted to arbitrary $\mathcal{L}$-expressions. This is equivalent to assuming a well-founded ordering on expressions of $\mathcal{L}$. For any reasonable definition such a well-founded ordering exists. Thus, although it is not difficult to imagine formulae ϕ^σ such that $\prec_0$ is not well-founded, we assume that the ϕ^σ are chosen in such a way that it is possible to lift the semantics of $\mathcal{L}$-primitives to all $\mathcal{L}$-expressions, i.e., $\prec_0$ is well-founded.

Recall that S^0 denotes the set of $\mathcal{L}$-open sentences that define the $\mathcal{L}$-connectives. A semantic specification S is *well-defined* iff

(wd1) $\forall S^0, \forall S^b \models \forall S$,
(wd2) the ordering $\prec$ is well-founded, and
(wd3) $\forall S^0, S^b{\restriction}\mathsf{sub}_{\prec}(\sigma(\overline{p})) \models_c \forall\overline{x}\Big(\Big(\bigwedge \Phi_{+}^{\sigma(\overline{p})} \rightarrow \phi^\sigma(\overline{p}, \overline{x})\Big) \wedge \Big(\phi^\sigma(\overline{p}, \overline{x}) \rightarrow \bigvee \Phi_{-}^{\sigma(\overline{p})}\Big)\Big).$

According to this definition, a well-defined semantics S is equivalent to $S^0 \cup S^b$ modulo the background theory S^b. This is ensured by condition (wd1) and the assumption that S defines all $\mathcal{L}$-connectives in S^0. Through condition (wd2), S imposes its own inductive structure on $\mathcal{L}$-expressions. Condition (wd3) specifies a correlation between S and S^0 on instances of $\mathcal{L}$-expressions. It can be seen that $S^0 \cup S^b$ is a well-defined semantic specification itself.

A *(propositional) logic* L over the language $\mathcal{L}$ is a subset of concepts in $\mathcal{L}$ which is closed under arbitrary substitutions of variables with expressions of the same sorts. A logic L is *first-order definable* iff there is a semantic specification S_L such that L coincides with the set of all concepts that are valid in all $\mathcal{L}$-structures satisfying $\forall S_L$, i.e., $L = \{C \in \mathcal{L}^1 \mid \forall S_L \models_c \forall x\, \nu_1(C, x)\}$.

For a fixed semantic specification S_L of logic L, if $\mathcal{I}$ is an $\mathcal{L}$-structure satisfying S_L then by definition $\mathcal{I}$ is a *model of* L or simply a *L-model* (with respect to S_L).

The following are examples of first-order definable logics, which all have a normalised semantic specification according to the above definitions: most description logics, including $\mathcal{ALCO}$, $\mathcal{ALBO}$ [5], $\mathcal{SHOIQ}$ [1], most propositional modal logics, including K, $\mathsf{S4}$, $\mathsf{KD45}$, propositional intuitionistic logic [3], and the logic of metric and topology [2].

3 Synthesising a Tableau Calculus

A *tableau calculus* is a set of tableau inference rules. A *tableau inference rule* is a rule of the form $X/X_1 \mid \cdots \mid X_m$, where both the numerator X and all denominators X_i ($m \geq 0$) are finite sets of negated or unnegated atomic formulae in the language $\mathsf{FO}(\mathcal{L})$. The formulae in the numerator are called *premises*, while the formulae in the denominators are called *conclusions*. The numerator and all the denominators are non-empty, but m may be zero, in which case the rule is a *closure rule* and is usually written $X/\bot$. If $m > 1$, the rule is a branching rule.

Inference steps are performed as usual. A rule is applied to a set of (ground) literals in a branch of a tableau derivation, if the literals are instances of the premises of the rule. Then, in the case of a non-branching rule, the corresponding (ground) instances of the conclusions of the rule are added to the branch. In the case of a branching rule the branch is split into several branches and the corresponding (ground) instances of the conclusions are added to each branch.

Let T denote a tableau calculus and C a concept. We take an arbitrary constant a of the designated sort which does not occur in the rules of T. We denote by $T(C)$ a finished tableau derivation built by starting with the formula $\nu_1(C, a)$ as input and applying the rules of T. That is, all branches in the tableau derivation are fully expanded and all applicable rules of T have been applied in $T(C)$. As usual we assume that all the rules of the calculus are applied *non-deterministically in a tableau derivation.* A branch of a tableau derivation is *closed* if a closure rule has been applied, otherwise the branch is called *open.* The tableau derivation $T(C)$ is *closed* if all its branches are closed and $T(C)$ is *open* otherwise. The calculus T is *sound* (for L) iff for any concept C, each $T(C)$ is open whenever C is satisfiable in an L-model. T is *complete* iff for any unsatisfiable concept C there is a $T(C)$ which is closed. T is said to be *terminating* if every finished open tableau derivation in T has a finite open branch.

Let L be a first-order definable propositional logic over $\mathcal{L}$ and S_L a well-defined semantic specification of L. We now describe how tableau rules can be synthesised from S_L. If S_L is not already normalised we first normalise it. Thus assume $S_L = S_L^+ \cup S_L^- \cup S_L^b$. Now take a positive specification ξ_+^E in S_L^+. Eliminate quantifiers using Skolemisation and equivalently rewrite ξ_+^E into the following implicational form

$$\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n) \rightarrow \bigvee_{j=1}^{J} \bigwedge_{k=1}^{K_j} \psi_{jk},$$

where each ψ_{jk} denotes a literal. This is always possible. The implication is now turned into the rule:

$$\rho_+(\xi_+^E) \overset{\text{def}}{=} \frac{\nu_n(E(p_1, \ldots, p_m), x_1, \ldots, x_n), \quad y_1 \approx y_1, \quad \ldots, \quad y_s \approx y_s}{\psi_{11}, \quad \ldots, \quad \psi_{1K_1} \mid \cdots \mid \psi_{J1}, \quad \ldots, \quad \psi_{JK_J}},$$

where $y_1, \ldots, y_s$ denote the free variables in ψ_{jk} which are not among the variables $x_1, \ldots, x_n$. Essentially, the antecedent of the implication has become the main premise in the nominator and the succedent was appropriately turned into the denominators of the rule. The purpose of the equations $y_i \approx y_i$ is domain predication. We say the rule *corresponds* to ξ^E_+. Analogously a tableau rule is be generated for each negative specification ξ^E_- in S^-_L. The contrapositive of ξ^E_- is equivalently rewritten to Skolemised implicational form

$$\neg\nu_n(E(p_1,\dots,p_m),x_1,\dots,x_n) \to \bigvee_{j=1}^{J}\bigwedge_{k=1}^{K_j}\psi_{jk},$$

where each ψ_{jk} denotes a literal, and the corresponding rules have the form

$$\rho_-(\xi^E_-) \overset{\text{def}}{=} \frac{\neg\nu_n(E(p_1,\dots,p_m),x_1,\dots,x_n),\quad y_1\approx y_1,\quad \dots,\quad y_s\approx y_s}{\psi_{11},\quad\dots,\quad\psi_{1K_1}\mid\cdots\mid\psi_{J1},\quad\dots,\quad\psi_{JK_J}}.$$

We refer to the rules $\rho_+(\xi^E_+)$ and $\rho_-(\xi^E_-)$ as *decomposition rules.*

For example, the generated decomposition rules for the existential restriction operator in the description logic $\mathcal{ALC}$ are:

$$\frac{\nu_1(\exists r.p,x)}{\nu_2(r,x,f(p,x)),\quad \nu_1(p,f(p,x))},\qquad\qquad \frac{\neg\nu_1(\exists r.p,x),\quad y\approx y}{\neg\nu_2(r,x,y)\mid\neg\nu_1(p,y)}.$$

These are not the familiar rules used in standard description logic tableau systems, but in Section 5 we see how to get those.

The sentences in the background theory of S_L are turned into rules by first equivalently transforming them into Skolemised disjunctive normal form. More specifically, let ξ be an arbitrary sentence in S^b_L. It is first rewritten to

$$(**)\qquad\qquad \bigvee_{j=1}^{J}\bigwedge_{k=1}^{K_j}\psi_{jk},$$

where each ψ_{jk} denotes a literal, and is then turned into the corresponding rule, namely

$$\rho(\xi) \overset{\text{def}}{=} \frac{p_1\approx p_1,\quad\dots,\quad p_m\approx p_m,\quad x_1\approx x_1,\quad\dots,\quad x_n\approx x_n}{\psi_{11},\quad\dots,\quad\psi_{1K_1}\mid\cdots\mid\psi_{J1},\quad\dots,\quad\psi_{JK_J}}.$$

The $p_1,\dots,p_m,x_1,\dots,x_n$ are the variables that are free in $(**)$. Rules corresponding to sentences in S^b_L are called *theory rules.*

We use T_L to denote the generated calculus. In summary, it consists of these rules.

(t1) The decomposition rules $\rho^\sigma_+(\xi)$ and $\rho^\sigma_-(\eta)$ corresponding to all positive specifications ξ in S^+_L and all negative specifications η in S^-_L.

(t2) The theory rules $\rho(\zeta)$ corresponding to all sentences ζ in the background theory S^b_L.

(t3) The *closure rules* (for every $n = 1,\dots,N$, and every constant predicate symbol P in S_L):

$$\frac{\nu_n(p,\overline{x}),\quad\neg\nu_n(p,\overline{x})}{\bot},\qquad\qquad\frac{P(\overline{x}),\quad\neg P(\overline{x})}{\bot}.$$

4 Ensuring Soundness and Constructive Completeness

It is possible to prove that every rule of the generated calculus T_L preserves satisfiability of $\mathit{FO}(\mathcal{L})$-formulae. That is, if all premises of a rule are true in an L-model $\mathcal{I}$ (under a canonical valuation) then the conclusions of some branch are also true. This is not difficult to see because the definitions of the rules mimic the specified semantics. Hence:

Theorem 1 (Soundness). *T_L is sound for L, i.e., for every concept C satisfiable in an L-model, any finished tableau derivation $T_L(C)$ is open.*

Now, we prove constructive completeness of T_L. Let $\mathcal{B}$ denote an arbitrary branch in a T_L-tableau derivation. We define the following relation $\sim_{\mathcal{B}}$ with respect to $\mathcal{B}$: $t \sim_{\mathcal{B}} t' \overset{\mathsf{def}}{\iff} t \approx t' \in \mathcal{B}$, for any ground terms t and t' of the designated sort $N+1$ in $\mathcal{B}$. Let $\|t\| \overset{\mathsf{def}}{=} \{t' \mid t \sim_{\mathcal{B}} t'\}$ be the equivalence class of an element t. The presence of the rules generated from the equality axioms ensure that $\sim_{\mathcal{B}}$ is a congruence relation on all designated ground terms in $\mathcal{B}$.

We say a model $\mathcal{I}$, under a (canonical) valuation ι, *reflects* an expression E occurring in a branch $\mathcal{B}$ iff for every ground terms $t_1, \ldots, t_n$ we have that

– $(E, \iota(t_1), \ldots, \iota(t_n)) \in \nu_n^{\mathcal{I}}$ whenever $\nu_n(E, t_1, \ldots, t_n) \in \mathcal{B}$, and
– $(E, \iota(t_1), \ldots, \iota(t_n)) \notin \nu_n^{\mathcal{I}}$ whenever $\neg\nu_n(E, t_1, \ldots, t_n) \in \mathcal{B}$.

Similarly, $\mathcal{I}$ *reflects* predicate constant P from $\mathcal{B}$ under a (canonical) valuation ι in $\mathcal{I}$ iff for every ground terms $t_1, \ldots, t_n$ we have that

– $(\iota(t_1), \ldots, \iota(t_n)) \in P^{\mathcal{I}}$ whenever $P(t_1, \ldots, t_n) \in \mathcal{B}$, and
– $(\iota(t_1), \ldots, \iota(t_n)) \notin P^{\mathcal{I}}$ whenever $\neg P(t_1, \ldots, t_n) \in \mathcal{B}$.

A model $\mathcal{I}$ *reflects* branch $\mathcal{B}$ under a valuation ι if $\mathcal{I}$ reflects all predicate constants and expressions occurring in $\mathcal{B}$ under ι.

A tableau calculus T_L is said to be *constructively complete* (for L) iff for any given concept C that is satisfiable, if $\mathcal{B}$ is an open branch in the tableau derivation $T_L(C)$ then there is an L-model $\mathcal{I}$ such that:

(m1) The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is the set of the equivalence classes $\|t\|$ for each ground term t occuring in $\mathcal{B}$.
(m2) $\mathcal{I}$ reflects $\mathcal{B}$ under the *canonical projection valuation* π defined by $\pi(t) \overset{\mathsf{def}}{=} \|t\|$, for every ground term t occuring in $\mathcal{B}$.

It is clear that if T_L is constructively complete then T_L is complete for L.

Suppose now that S_L is a semantic specification and $\prec_0$ is a well-founded ordering on $\mathcal{L}$-expressions induced by the set S_L^0 of the definitions of the connectives of the form $(*)$ with respect to S_L.

Let $\mathcal{B}$ be an open branch in a finished tableau derivation in T_L. Define interpretations of predicate symbols in $\mathcal{I}(\mathcal{B})$ by induction on $\prec_0$ as follows:

– $P^{\mathcal{I}(\mathcal{B})} \overset{\mathsf{def}}{=} \{(\|t_1\|, \ldots, \|t_n\|) \mid P(t_1, \ldots, t_n) \in \mathcal{B}\}$, for every n-ary constant predicate symbol P in S_L.

– For every $n = 1, \ldots, N$ the interpretation $\nu_n^{\mathcal{I}(\mathcal{B})}$ of the ν_n symbols is defined as the smallest subset of $\mathcal{L}^n \times (\Delta^{\mathcal{I}(\mathcal{B})})^n$ satisfying both $(p, \|t_1\|, \ldots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})} \iff \nu_n(p, t_1, \ldots, t_n) \in \mathcal{B}$ and $(\sigma(E_1, \ldots, E_m), \|t_1\|, \ldots, \|t_n\|) \in \nu_n^{\mathcal{I}(\mathcal{B})} \iff \mathcal{I}(\mathcal{B}) \models_c \phi^\sigma(E_1, \ldots, E_m, \|t_1\|, \ldots, \|t_n\|)$, for every variable or constant p of the sort n, every connective σ, and any expressions $E_1, \ldots, E_m$.

A consequence of the definition of $\mathcal{I}(\mathcal{B})$ is that the definitions of the connectives are valid in $\mathcal{I}(\mathcal{B})$, i.e., we have $\mathcal{I}(\mathcal{B}) \models \forall S_L^0$.

It can be proved that $\mathcal{I}(\mathcal{B})$ reflects the branch $\mathcal{B}$ (under the valuation π) by induction on the well-founded ordering $\prec$. As a consequence we obtain constructive completeness.

Theorem 2 (Constructive completeness). *T_L is constructively complete.*

5 Refining the Synthesised Calculus

In order to get inference rules that have better properties, in this section we introduce two refinements.

The first refinement reduces the number of branches of a rule by constraining the rule with additional premises rather than deriving new conclusions. Suppose $r \overset{\text{def}}{=} X/X_1 \mid \cdots \mid X_m$ is a tableau rule of a sound and constructively complete tableau calculus T_L. For some $i \in \{1, \ldots, m\}$ suppose $X_i = \{\psi_1, \ldots, \psi_k\}$. Without loss of generality we can assume that $i = 1$. Consider the rules r_j with $j = 1, \ldots, k$ defined by

$$r_j \overset{\text{def}}{=} \frac{X \cup \{\sim\psi_j\}}{X_2 \mid \cdots \mid X_m}.$$

Note that we can drop any domain predication equalities from the numerator when they are not necessary. Let T'_L be the tableau calculus obtained from T_L by replacing rule r by the rules $r_1, \ldots, r_k$. It is clear that T'_L is sound. In general, T'_L is not constructively complete. However the following theorem is true.

Theorem 3. *Let $\mathcal{B}$ be an open branch in a T'_L-tableau. Assume that for every set Y of $\mathcal{L}$-expressions the following holds.*

If all expressions from Y are reflected in $\mathcal{I}(\mathcal{B})$ then for every $E_1, \ldots, E_l \in Y$,

(†) $X(E_1, \ldots, E_l, t_1, \ldots, t_n) \subseteq \mathcal{B}$ *implies*

$$\mathcal{I}(\mathcal{B}) \models X_i(E_1, \ldots, E_l, \|t_1\|, \ldots, \|t_n\|) \text{ for some } i = 1, \ldots, m.$$

Then, $\mathcal{B}$ is reflected in $\mathcal{I}(\mathcal{B})$.

Corollary 1. *If the condition of Theorem 3 holds for every open branch $\mathcal{B}$ of any T'_L-tableau then T'_L is constructively complete.*

Generalising this refinement to moving more than one conclusion up to the numerator is not difficult. The formulation of Theorem 3 does not change then.

Consider the generated rule for negative occurrences of the existential restriction operator given on p. 8. In most description logics it can be transformed to the more often seen rule:

$$\frac{\neg\nu_1(\exists r.p, x), \quad \nu_2(r, x, y)}{\neg\nu_1(p, y)}.$$

In order to preserve constructive completeness, the following condition is usually proved by induction on $\prec$. This, in turn, inductively implies condition (†).

$$\text{If } \neg\nu_1(\exists E.F, t) \in \mathcal{B} \text{ and } \mathcal{I}(\mathcal{B}) \models \nu_2(E, t, t') \text{ then } \neg\nu_1(F, t') \in \mathcal{B}.$$

Another example of a generated rule and a refinement are (for transitive R):

$$\frac{x \approx x, \quad y \approx y, \quad z \approx z}{\neg R(x,y) \mid \neg R(y,z) \mid R(x,z)}, \qquad\qquad \frac{R(x,y), \quad R(y,z)}{R(x,z)}.$$

Condition (†) holds in this case since, by definition of $\mathcal{I}(\mathcal{B})$, $\mathcal{I}(\mathcal{B})$ reflects all atomic predicate constants in the branch $\mathcal{B}$.

The second refinement we describe exploits the expressivity of the logic. Suppose that the language $\mathcal{L}$ of the logic L is expressive enough to represent its own semantics. That is, assume that for every $n = 0, \ldots, N$ and every n-ary predicate constant P occuring in S_L, there are concepts $C_n^+(p, \ell_1, \ldots, \ell_n)$, $C_n^-(p, \ell_1, \ldots, \ell_n)$, $D_P^+(\ell_1, \ldots, \ell_n)$, and $D_P^-(\ell_1, \ldots, \ell_n)$, depending on variable p of sort n and variables $\ell_1, \ldots, \ell_n$ of sort 0, such that

$$\begin{aligned}
&\forall S_L \models \forall x \left(\nu_1(C_n^+(p, \ell_1, \ldots, \ell_n), x) \rightarrow \nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right),\\
&\forall S_L \models \forall x \left(\nu_1(C_n^-(p, \ell_1, \ldots, \ell_n), x) \rightarrow \neg\nu_n(p, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right),\\
&\forall S_L \models \forall x \left(\nu_1(D_P^+(\ell_1, \ldots, \ell_n), x) \rightarrow P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right),\\
&\forall S_L \models \forall x \left(\nu_1(D_P^-(\ell_1, \ldots, \ell_n), x) \rightarrow \neg P(\nu_0(\ell_1), \ldots, \nu_0(\ell_n))\right).
\end{aligned}$$

In this case we are able to express all tableau rules for L in the language $\mathcal{L}$ itself. We only need to replace every positive occurrence of $\nu_n(E, x_1, \ldots, x_n)$ in T_L with $C_n^+(E, \ell_1, \ldots, \ell_n)$, every (negative) occurrence of $\neg\nu_n(E, x_1, \ldots, x_n)$ in T_L with $C_n^-(E, \ell_1, \ldots, \ell_n)$, and, similarly, all the predicate constants P need to be replaced with occurrences of D_P^+ or D_P^- depending on the polarity of P. In fact, the sort $N+1$ of $\mathit{FO}(\mathcal{L})$ can be reflected in the sort 0.

A slight difficulty is caused by Skolem functions in $\mathit{FO}(\mathcal{L})$ occurring in the tableau calculus, since for them there could be no corresponding function symbols in $\mathcal{L}$. It can be solved by introducing new connectives f_g into the language $\mathcal{L}$ for every (Skolem) function and constant g of $\mathit{FO}(\mathcal{L})$ so that for every $(p_1, \ldots, p_m, \ell_1, \ldots, \ell_n)$, $f_g(p_1, \ldots, p_m, \ell_1, \ldots, \ell_n)$ is a term of the sort 0 and its semantics is defined by $\nu_0(f_g(\overline{p}, \ell_1, \ldots, \ell_n)) \overset{\text{def}}{=} g(\overline{p}, \nu_0(\ell_1), \ldots, \nu_0(\ell_n))$. An alternative is to introduce unique, new individual names (for every $p_1, \ldots, p_m$, $\ell_1, \ldots, \ell_n$) instead of new connectives.

Connective definitions
$\forall x\ \big(\nu_1(\bot, x) \equiv \bot\big)$
$\forall x\ \big(\nu_1(p_1 \wedge p_2, x) \equiv \nu_1(p_1, x) \wedge \nu_1(p_2, x)\big)$
$\forall x\ \big(\nu_1(p_1 \vee p_2, x) \equiv \nu_1(p_1, x) \vee \nu_1(p_2, x)\big)$
$\forall x\ \big(\nu_1(p_1 \rightarrow p_2, x) \equiv \forall y\ \big(R(x,y) \rightarrow (\nu_1(p_1, y) \rightarrow \nu_1(p_2, y))\big)\big)$

Background theory of a partial ordering R and ν_1
$\forall x\ R(x,x)$
$\forall x \forall y\ (R(x,y) \wedge R(y,x) \rightarrow x \approx y)$
$\forall x \forall y \forall z\ (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$
$\forall x \forall y\ \big(\nu_1(p,x) \wedge R(x,y) \rightarrow \nu_1(p,y)\big)$

Figure 2. Specification of semantics of intuitionistic logic.

For example, in the description logic $\mathcal{ALCO}$ with full support of individuals, we can set (for any atomic role r and individual equality):

$$C_2^+(r, \ell_1, \ell_2) \overset{\text{def}}{=} \ell_1 : \exists r.\{\ell_2\}, \qquad D_{\approx}^+(\ell_1, \ell_2) \overset{\text{def}}{=} \ell_1 : \{\ell_2\},$$
$$C_2^-(r, \ell_1, \ell_2) \overset{\text{def}}{=} \ell_1 : \neg\exists r.\{\ell_2\}, \qquad D_{\approx}^-(\ell_1, \ell_2) \overset{\text{def}}{=} \ell_1 : \neg\{\ell_2\}.$$

Thus, the language of the tableau calculus can be significantly simplified. For instance, the (refined) rules for the existential restriction operator become:

$$\frac{\ell : \exists r.p}{\ell : \exists r.\{f(r,p,\ell)\}, \quad f(r,p,\ell) : p}, \qquad \frac{\ell : \neg\exists r.p, \quad \ell : \exists r.\{\ell'\}}{\ell' : \neg p}.$$

6 Synthesising Tableaux for Intuitionistic Logic

Intuitionistic logic is an example of a logic where ν_n cannot be expressed in the language of the logic. It also provides an example of a logics for which a background theory is an essential part of the definition of the semantics.

The language of intuitionistic logic is a one-sorted language defined over a countable set of propositional symbols $p_1^1, p_2^1, \ldots$, and the standard connectives are $\rightarrow, \vee, \wedge, \bot$. The semantic specification in $FO(\mathcal{L})$ is given in Figure 2 (cf. [3]). R is the designated predicate symbol representing the partial order in the background theory. For intuitionistic logic the orderings $\prec_0$ and $\prec$ coincide. The ordering $\prec$ on subexpressions induced by the semantic definition of the connectives is the smallest ordering satisfying: $E_1 \prec E_1 \sigma E_2$ and $E_2 \prec E_1 \sigma E_2$, for each $\sigma \in \{\rightarrow, \vee, \wedge\}$ and all intuitionistic formulae E_1 and E_2.

The tableau rules generated by our approach are those listed in Figure 3. Together with the equality rules, they form a calculus, which is sound and constructively complete for propositional intuitionistic logic. This is an immediate consequence of Theorems 1 and 2. Refining the generated rules yields the rules listed in Figure 4. Using Theorem 3 we conclude that these rules together with suitably refined equality rules provide a sound and constructively complete tableau calculus for intuitionistic logic.

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\nu_1(p_1 \wedge p_2, x)}{\nu_1(p_1, x), \quad \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \wedge p_2, x)}{\neg\nu_1(p_1, x) \mid \neg\nu_1(p_2, x)}$$

$$\frac{\neg\nu_1(\bot, x)}{\neg\bot} \qquad \frac{\nu_1(p_1 \vee p_2, x)}{\nu_1(p_1, x) \mid \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \vee p_2, x)}{\neg\nu_1(p_1, x), \quad \neg\nu_1(p_2, x)}$$

$$\frac{\nu_1(p_1 \rightarrow p_2, x)}{\neg R(x, y) \mid \neg\nu_1(p_1, y) \mid \nu_1(p_2, y)}$$

$$\frac{\neg\nu_1(p_1 \rightarrow p_2, x)}{R(x, f(p_1, p_2, x)), \quad \nu_1(p_1, f(p_1, p_2, x)), \quad \neg\nu_1(p_2, f(p_1, p_2, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x, x)} \qquad \frac{x \approx x, \quad y \approx y}{\neg R(x, y) \mid \neg R(y, x) \mid x \approx y} \qquad \frac{x \approx x, \quad y \approx y, \quad z \approx z}{\neg R(x, y) \mid \neg R(y, z) \mid R(x, z)}$$

$$\frac{p \approx p, \quad x \approx x, \quad y \approx y}{\neg\nu_1(p, x) \mid \neg R(x, y) \mid \nu_1(p, y)}$$

Closure rules:

$$\frac{\nu_1(p, x), \quad \neg\nu_1(p, x)}{\bot} \qquad \frac{R(x, y), \quad \neg R(x, y)}{\bot}$$

Figure 3. Generated tableau rules for intuitionistic logic.

A terminating tableau calculus is obtained if the calculus is enhanced with the blocking mechanism of [5,6]. This follows from the results in [6], the soundness and constructive completeness of the calculus, the subexpression property and the fact that intuitionistic logic admits finite filtrations. The calculus can be turned into a deterministic decision procedure using breadth-first search or depth-first search, as we showed in [6].

7 Discussion and Conclusions

The method introduced in this paper automatically produces a sound and constructively complete tableau calculus from a semantic first-order specification of a many-sorted logic. The method is directly applicable to many non-classical logics and covers many types of ground tableau calculi commonly found in the literature. These include two types of tableau calculi for relations satisfying extra theory conditions which can be accommodated either by structural rules or propagation rules.

The results of the paper can be regarded as a mathematical formalisation and generalisation of tableau development methodologies. Our formalisation is based on, and provides the basis for, the implementation of tableau decision procedures for modal and description logics in the MetTeL system [8]. The formalisation separates the creative part of tableau calculus development, which needs to be done by a human developer, and the automatic part of the development process,

Decomposition rules:

$$\frac{\nu_1(\bot, x)}{\bot} \qquad \frac{\nu_1(p_1 \wedge p_2, x)}{\nu_1(p_1, x), \quad \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \wedge p_2, x)}{\neg\nu_1(p_1, x) \mid \neg\nu_1(p_2, x)}$$

$$\frac{\nu_1(p_1 \vee p_2, x)}{\nu_1(p_1, x) \mid \nu_1(p_2, x)} \qquad \frac{\neg\nu_1(p_1 \vee p_2, x)}{\neg\nu_1(p_1, x), \quad \neg\nu_1(p_2, x)} \qquad \frac{\nu_1(p_1 \rightarrow p_2, x), \quad R(x,y), \quad \nu_1(p_1, y)}{\nu_1(p_2, y)}$$

$$\frac{\neg\nu_1(p_1 \rightarrow p_2, x)}{R(x, f(p_1, p_2, x)), \quad \nu_1(p_1, f(p_1, p_2, x)), \quad \neg\nu_1(p_2, f(p_1, p_2, x))}$$

Theory rules:

$$\frac{x \approx x}{R(x,x)} \qquad \frac{R(x,y), \quad R(y,x)}{x \approx y} \qquad \frac{R(x,y), \quad R(y,z)}{R(x,z)} \qquad \frac{\nu_1(p,x), \quad R(x,y)}{\nu_1(p,y)}$$

Closure rules:

$$\frac{\nu_1(p,x), \quad \neg\nu_1(p,x)}{\bot}$$

Figure 4. Refined tableau rules for intuitionistic logic.

which can be left to an automated (currently first-order) prover and an automated tableau synthesiser. The creative part is the specification of the logic so that the conditions of well-foundness of the orderings $\prec_0$ and $\prec$ hold. The automatic part deals with verification of the first-order conditions (wd1) and (wd3), and the generation of tableau rules from the (well-defined) semantics provided by the developer. Then, the developer can transform the generated rules to refined form by applying Theorem 3. Refinements are crucial for the success of the method. With the described refinements the calculi that the method generates are equivalent (modulo inessential variations) to the calculi common for intuitionistic logic, $\mathcal{ALCO}$, most other first-order definable description logics, and the common modal logics such as S4 and S5, for example.

For common modal and description logics conditions (wd1) and (wd3) are simple to check, even trivial in many cases. In fact, a developer usually implicitly formalises the logic's semantics S in such way that $S = S^0 \cup S^b$. This is the case for almost all of known logics. If the specification of the semantics satisfies $S = S^0 \cup S^b$ then conditions (wd1) and (wd3) hold trivially and the orderings $\prec_0$ and $\prec$ coincide. This means the ordering used for the specification of the semantics of the logical connectives (which is usually well-founded), is enough for tableau synthesis.

This paper also presents a general method for proving (constructive) completeness of tableau calculi. In addition, the generated rules can be transformed to an optimal form provided that the special condition (†) has been proven by induction on the ordering $\prec$ for the refined calculus.

With enough expressivity for representing the basics of the semantics within the logic it is possible to simplify the language of the tableau. In this case, the obtained calculus is similar to tableau calculi for description logics with full support of individuals, hybrid modal logic, and labelled tableau calculi. Otherwise, the calculus has the same flavour as the standard tableau calculus for intuitionistic logic, where every node of a tableau is characterised by two complementary sets of true and false formulae (concepts).

As a case study we considered tableau synthesis for propositional intuitionistic logic. We believe the approach is also applicable to most known, first-order definable modal and description logics. Non first-order translatable logics such as propositional dynamic logic are currently beyond the scope of the method.

The tableau calculi generated are Smullyan-type tableau calculi, i.e., ground semantic tableau calculi. We believe that other types of tableau calculi can be generated using the same techniques. Exploiting the known relationships to other deduction methods and the underlying ideas of [4] we expect synthesis of non-tableau approaches is possible as well, but this is future work. In [4] we have shown that it is possible to synthesise tableau calculi for modal logics by translation to first-order logic combined with first-order resolution. In this framework the semantic specification of a logic is transformed into clausal form and then a set of inference rules. Soundness and completeness of the generated calculus follows from the soundness and completeness of the simulating resolution refinement used. This approach has several advantages, but in this paper we have taken a different, more direct approach. Rather than proceeding via simulation by resolution we have shown that tableau rules can be generated directly from the specification of the logic.

Our future goal is to further reduce human involvement in the development of calculi by finding appropriate automatically verifiable conditions for optimal calculi to be generated.

References

1. I. Horrocks and U. Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *J. Automat. Reasoning*, 39(3):249–276, 2007.
2. U. Hustadt, D. Tishkovsky, F. Wolter, and M. Zakharyaschev. Automated reasoning about metric and topology (System description). In *JELIA06*, vol. 4160 of *LNAI*, pp. 490–493. Springer, 2006.
3. S. A. Kripke. Semantical analysis of intuitionistic logic I. In *Formal Systems and Recursive Functions*, pp. 92–130. North-Holland, 1965.
4. R. A. Schmidt. Developing modal tableaux and resolution methods via first-order resolution. In *Advances in Modal Logic, Volume 6*, pp. 1–26. College Publ., 2006.
5. R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *ISWC07*, vol. 4825 of *LNCS*, pp. 438–451. Springer, 2007.
6. R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In *IJCAR08*, vol. 5195 of *LNCS*, pp. 194–209. Springer, 2008.
7. R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi, 2009. `http://www.cs.man.ac.uk/~dmitry/papers/astc2009.pdf`.
8. D. Tishkovsky. MetTeL system. `http://www.cs.man.ac.uk/~dmitry/implementations/MetTeL/`.