Universität Tübingen, Seminar für Sprachwissenschaft
Research Apprenticeship *Language Structure*
Supervisor: Dr. Johannes Dellert

Automated Model Checking, Theorem Proving and Model Generation for Linguistic Applications

Natalie Clarius

First version
Date of submission: December 25, 2020

Contents

1 Introduction

The goal of this research apprenticeship was to investigate methods of model checking, theorem proving and automated model generation for applications in linguistics. The main result of our work is an implementation of both model checking using recursive semantic evaluation, and tableau-based methods for theorem proving and model generation.

The purpose of this work is twofold. On the one hand, our implementation is a research tool to investigate linguistic phenomena such as ambiguity and intensional contexts directly. In order to achieve this, the implementation considers several logics: Classical propositional and first-order logic, propositional modal logic, first-order modal logic with constant domains and first-order modal logic with varying domains. On the other hand, our work will serve as a didactic foundation for linguistic studies, in particular the possibility on linking basic programming techniques to formal methods as taught in undergraduate linguistics programs, and we would like to extend this in future linguistics courses.

The objective of this paper is to discuss to which degree formal computational models are suitable for natural language analysis, sampling the relevant literature in this area. After a brief summary of the theoretical foundation and implementation in Section 2, we will discuss a selection of linguistic phenomena with respect to their formal and computational tractability in Section 3. We conclude in Section 4 with an outlook on possible extensions and usage scenarios, with respect to both teaching and research.

Our program, named pyPL (for "*py*thon *P*redicate *L*ogic"), is open source software and available to the community on GitHub via https://github.com/nclarius/pyPL.

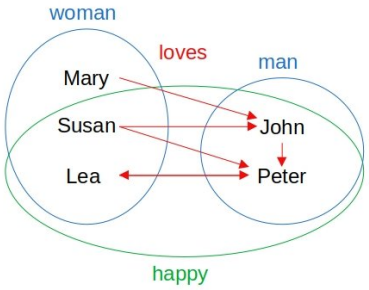
2 Theoretical foundation and implementation

2.1 Model checking with recursive evaluation

Situations described by natural language discourse can formally be modelled by structures. In first-order logic, a *structure* is a pair of a *domain of discourse* $\mathcal{D}$ and an *interpretation* $\mathcal{I}$ of the non-logical symbols of the language, mapping individual constants to objects of the domain and n-ary predicate symbols to n-ary relations over the domain. A *model* of a sentence is a structure in which that sentence is true; a model of a set of sentences is a structure in which all of the sentences are true.

Model checking is the problem of verifying whether a structure is a model of a given sentence. In a broader sense, model checking can also encompass the computation of the denotation of an arbitrary expression, for instance the referent of an individual term, as opposed to only truth values of sentences.

For instance, consider the following situation, depicted diagrammatically with circles signifying properties and arrows signifying relations:



This situation is represented by the structure $\mathcal{S}_1 = \langle \mathcal{D}_1, \mathcal{I}_1 \rangle$ with

$$
\begin{aligned}
\mathcal{D}_1 &= \{\text{John}, \text{Lea}, \text{Mary}, \text{Peter}, \text{Susan}\} \\
\mathcal{I}_1 : \quad & \textit{lea} \mapsto \text{Lea} \\
& \textit{mary} \mapsto \text{Mary} \\
& \textit{susan} \mapsto \text{Susan} \\
& \textit{john} \mapsto \text{John} \\
& \textit{Peter} \mapsto \text{Peter} \\
& \textit{Happy} \mapsto \{\langle \text{John} \rangle, \langle \text{Lea} \rangle, \langle \text{Peter} \rangle, \langle \text{Susan} \rangle\} \\
& \textit{Woman} \mapsto \{\langle \text{Lea} \rangle, \langle \text{Mary} \rangle, \langle \text{Susan} \rangle\} \\
& \textit{Man} \mapsto \{\langle \text{John} \rangle, \langle \text{Peter} \rangle\} \\
& \textit{Love} \mapsto \{\langle \text{John}, \text{Peter} \rangle, \langle \text{Lea}, \text{Peter} \rangle, \langle \text{Mary}, \text{John} \rangle, \langle \text{Peter}, \text{Lea} \rangle, \\
& \qquad \langle \text{Susan}, \text{John} \rangle, \langle \text{Susan}, \text{Peter} \rangle\}
\end{aligned}
$$

Note that this structure is finite, since the domain of discourse contains only finitely many elements. Due to the nature of the implementation of structures in Python with discourses of the data type `Set`, pyPL can only treat structures with finitely many elements. While this is a substantial limitation from a theoretical point of view and makes the program unsuited for most applications in mathematics, for linguistic modelling it is usually sufficient to limit oneself to such finite structures.

Definitions of finite structures can be implemented as data structures in Python in a rather direct way: Non-logical symbols and real world objects are encoded as strings,

sets and tuples exist as primitive data structures in Python, and finite functions are most suitably implemented as dictionaries. A `Structure` type can then be specified to be instantiated with the appropriate components as fields. For instance, the above structure definition can be implemented in Python as follows:

```
s = "S1"
d = {"Mary", "Susan", "Lea", "John", "Peter"},
i = {"john": "John", "lea": "Lea", "mary": "Mary",
     "peter": "Peter", "susan": "Susan",
     "Happy": {("John",), ("Lea",), ("Peter",),
               ("Susan",)},
     "Woman": {("Mary", ), ("Susan",), ("Lea",)},
     "Man":   {("John",), ("Peter",)},
     "Love":  {("Mary", "John"), ("Susan", "Peter"),
               ("John", "Mary"), ("John", "Peter")}}
s1 = Structure(s, d, i)
```

As we can see, this is straightforward and thus makes it easy to convert examples discussed in introductory linguistics classes into a machine-readable format.

To represent formulas, a class for each logical operator is introduced, where the subexpressions the operator takes as arguments are class fields, and a formula object is instantiated with the respective subexpressions given as arguments to the constructor.

As an example, the class for atomic formulas (a predicate symbol applied to an appropriate number of individual terms) can be implemented as follows:

```
class Atom(Formula):
    """
    Atomic formula (predicate symbol applied to a number
    of terms).
    P(t1,...,tn)
    """"

    def __init__(self, predicate, terms):
        self.p = predicate  # the predicate symbol
        self.ts = terms     # the tuple of terms

    def denot(self, s, v, w):
        """The denotation of an atomic predication is
           true iff the tuple of the denotations of the
           terms is an element of the denotation of the
           predicate."""
        return tuple([t.denot(s, v, w) for t in self.ts])
               in self.p.denot(s, v, w)
```

The `denotation` method corresponds to the semantics clause of the respective expression. Here, the denotation of the atomic expression relative to a structure `s`, an assignment function `v` and a possible world `w` is composed recursively of the denotations of the predicate symbols and the terms.

As an example for a logical operator we consider material implication, which is implemented as follows:

```
class Implication(Formula):
    """
    Implication.
    (phi -> psi)
    """

    def __init__(self, formula1, formula2):
        self.phi = formula1  # the antecedent formula
        self.psi = formlua2  # the succedent formula

    def denot(self, s, v, w):
        """The denotation of an implicational formula is
           true iff phi is false or psi is true."""
        return not self.phi.denot(s, v, w) or
               self.psi.denot(s, v, w)
```

Again, the denotation is a function of the denotations of the formula's components, with truth conditions expressed in terms of boolean operations from the native Python inventory. This transparently illustrates the recursive evaluation of logical expressions, and users can precisely track the evaluation process by stepping though the function calls with debugging tools shipped with standard IDEs.

From a didactical point of view, existential quantification is an instructive example:

```
class Exists(Formula):
    """
    Existential quantification.
    Exists u phi
    """

    def __init__(self, variable, formula):
        self.u = variable   # the binding variable
        self.phi = formula  # the formula quantified over

    def denot(self, s, v, w):
        """ The denotation of an existential formula is
            true iff phi is true under at least one
            u-variant of v."""

        # iterate over all elements a of the domain
        for a in s.domain:
            # compute the assignment variant v' = v[u/a]
            v_ = v | {self.u: a}
            # check whether phi is true under v'
            if self.phi.denot(s, v_, w):

                # if yes, a witness as been found,
                # the existential statement is true,
                # and the check can be stopped (return)
                return True
```

```
        # if the and of the loop is reached
        # without a return,
        # then no witness has been found,
        # and the existential statement is false
        return False
```

The semantic iteration over all elements of the domain implemented in terms of a loop may help beginners learn formal methods and basic programming techniques in an integrative approach.

With this setup, complex formulas can be composed by recursively combining such expression objects. For instance, the formula

$$\phi := \forall x(\mathit{Woman}(x) \rightarrow \exists y(\mathit{Man}(y) \wedge \mathit{Love}(x, y)))$$

can be encoded as a Python object as follows:

```
f = Forall(Var("x"), Imp(Atm(Pred("Woman"), (Var("x"),)
    Exists(Var("y"), Conj(Atm(Pred("Man"), (Var("y"),)),
    Atm(Pred("Love"), (Var("x"), Var("y")))))))))
```

The pyPL interface provides a parser that allows the user to type formulas and structures with ordinary keyboard characters, which are then internally represented as described above.

Checking the semantic value of a linguistic expression such as

(1) Every woman loves a man.

in a structure as defined above, that is, computing

$$[\![\phi]\!]^{\mathcal{S}}$$

can be achieved by calling the `denotation` method on the formalized sentence with the structure given as an argument:

```
f.denot(s)
```

This will trigger a recursive evaluation for the semantic clauses, starting with the main operator (here: `Forall`) and eventually yielding a value of the appropriate type (here: the boolean `True`).

2.2 Theorem proving with validity tableaus

Going one step further, rather than asking whether a sentence is true in a particular structure, an interesting question is whether a sentence is true in all structures in which a given set of premises is true – this is the problem of semantics inference:

$$\psi_1, \ldots, \psi_n \models \phi \iff \begin{array}{l}\text{There exists no structure } \mathcal{S} \text{ such that} \\ [\![\psi_i]\!]^{\mathcal{S}} = \text{True for all } i \text{ and } [\![\phi]\!]^{\mathcal{S}} = \text{False.}\end{array}$$

As a special case, we have the problem of validity:

$$\models \phi \iff \text{There exists no structure } \mathcal{S} \text{ such that } [\![\phi]\!]^{\mathcal{S}} = \text{False.}$$

2.2.1 Tableaus

In order to tackle these problems algorithmically, we consider the calculus of analytic tableaus, which is sound and complete for non-modal and modal propositional and first-order logic. For theorem proving, we use the conventional variant which we call "validity tableaus". For model generation, we use a modified version called "satisfiability tableaus", explained in Section 2.3.

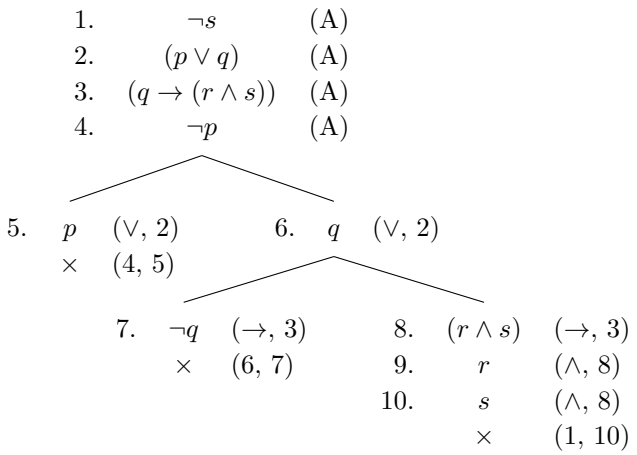
The calculus of analytic tableaus is a refutation calculus: A proof of an inference proceeds by assuming that the premises are true but the conclusion is false, and deriving a contradiction under these assumptions. For each logical operator there is one positive and one negative rule, corresponding to the operator occurring positively (for example, $\phi \wedge \psi$) and negatively (for example, $\neg(\phi \wedge \psi)$), respectively. The inference rules yield a tree structure systematically analyzing what must be the case if the assumptions are to hold, where branching represents a disjunction, and formulas in the same branch are read as being simultaneously true. Each branch, then, stands for one possibility of satisfying the assumptions. If both ϕ and $\neg\phi$ appear in a branch for some formula ϕ, then this way of attempting to invalidate the inference fails; the branch is then closed, which is indicated by the symbol $\times$. If no contradiction arises, then the branch depicts a way to satisfy the premises but not the conclusion and hence yields a counterexample. The branch is then open, which is indicated by the symbol $\bigcirc$. If after exhausting all applicable rules all branches of the tree are closed, then the tree is called closed, in which case there exists no counterexample and the tree proves that the inference is valid. If, on the other hand, at least one branch of the tree is open, then the tree is called open, and the inference is proven invalid since in this case there must exist a counterexample.

As an example, a closed validity tableau for the valid inference

$$p \vee q, q \rightarrow (r \wedge s)), \neg p \vDash s$$

is given as follows:

(2)

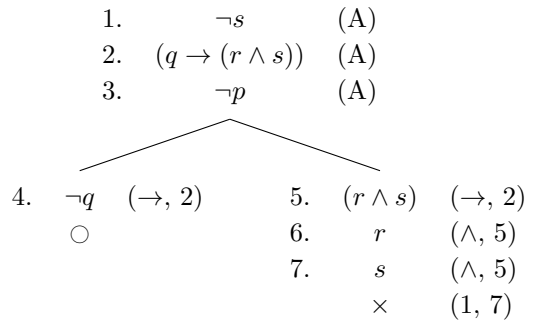


An open validity tableau for the invalid inference

$$q \rightarrow (r \wedge s)), \neg p \nvDash s$$

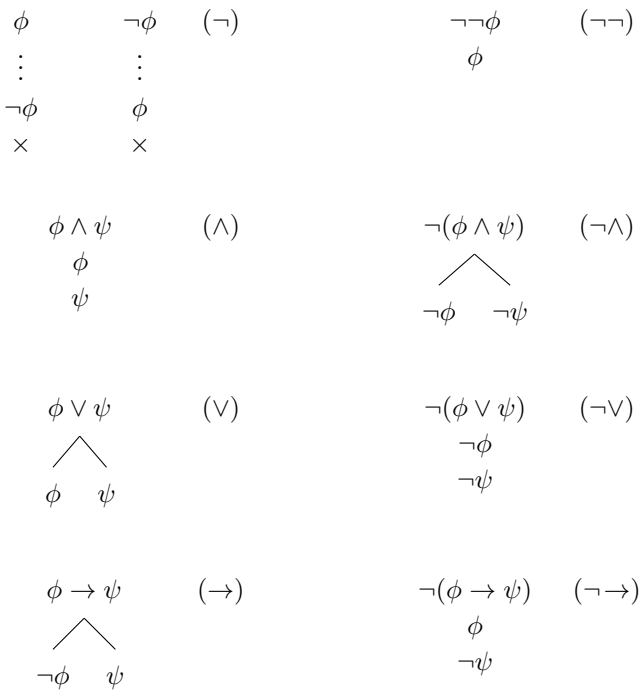
is the following:

(3)



2.2.2 Validity tableaus for propositional logic

The rules for the propositional connectives are given as follows (D'Agostino et al., 1999, p. 57):



We can distinguish between non-branching rules, also called α-rules, such as ($\wedge$), and branching rules, also called β-rules, such as ($\vee$). For example, the rule ($\wedge$) represents the idea that $\phi \wedge \psi$ is true iff ϕ is true and ψ is true; ($\vee$) expresses that $\phi \vee \psi$ is true if either ϕ is true or ψ is true.

2.2.3 Validity tableaus for first-order logic

The rules become slightly more complicated for first-order logic, because we have to consider eigenvariable conditions. Tableaus are extended to first-order logic by adding the following rules for the quantifiers (D'Agostino et al., 1999, p. 146):

$$\frac{\forall v\phi(v)}{\phi(c)}\ (\forall) \qquad c \text{ arbitrary} \qquad\qquad \frac{\neg\forall v\phi(v)}{\neg\phi(c)}\ (\neg\forall) \qquad c \text{ new}$$

$$\frac{\exists v\phi(v)}{\phi(c)}\ (\exists) \qquad c \text{ new} \qquad\qquad \frac{\neg\exists v\phi(v)}{\neg\phi(c)}\ (\neg\exists) \qquad c \text{ arbitrary}$$

Rules for quantifiers allow for the introduction of individual constants. We can distinguish two groups of rules, γ-rules ($\forall$) and ($\neg\exists$) and δ-rules ($\exists$) and ($\neg\forall$). γ-rules permit formulas to be instantiated with arbitrary constants, that is, constants that already occur in the branch or new constant symbols, whereas δ-rules require the constant introduced to be new. The rule ($\forall$) depicts the idea that if a formula holds for all objects, then it may be inferred for any individual; the rule ($\exists$) guarantees the existence of an individual with property ϕ, but a new constant has to be chosen in order to preserve generality by avoiding unjustified assumptions of identity to other individuals. From a semantic perspective, this condition excludes the possibility that contradictions arise merely due to additional properties imported by previous assumptions on another individual, rather than explicitly due to the properties of the object whose existence is asserted.

Quantifier rules can alternatively be formulated using free variables for universal formulas and Skolem functions for existential formulas, yielding the variant of free-variable tableaus (Dellert, 2010), which are not considered here.

2.2.4 Validity tableaus for modal logic

For the investigation of modal and intensional contexts, we also implemented the modal logic K. In order to reflect possible worlds and the accessibility relation between worlds, tableaus for modal logic introduce so-called signatures with which nodes in a tree are annotated; cf. (Fitting & Mendelsohn, 1998). A signature encodes simultaneously a possible world and an accessibility tuple: $\sigma.\tau$ expresses that world σ can access world $\sigma.\tau$. All formulas are read as relative to a signature; a contradiction only arises if two formulas $\phi, \neg\phi$ occur in the same branch with the same signature.

The tableau rules for the modal logic K are given as follows:

$$\frac{\sigma\ \Box\phi}{\sigma.n\ \phi}\ (\Box) \qquad \sigma.n \text{ old} \qquad\qquad \frac{\sigma\ \neg\Box\phi}{\sigma.n\ \neg\phi}\ (\neg\Box) \qquad \sigma.n \text{ new}$$

$$\frac{\sigma\ \diamond\phi}{\sigma.n\ \phi}\ (\Diamond) \qquad \sigma.n \text{ new} \qquad\qquad \frac{\sigma\ \neg\Diamond\phi}{\sigma.n\ \neg\phi}\ (\neg\Diamond) \qquad \sigma.n \text{ old}$$

Again, these rules can be divided into two groups: ν-rules ($\Box$) and ($\neg\Diamond$) and μ-rules ($\Diamond$) and ($\neg\Box$). The former require the signature $\sigma.n$ to already occur in the branch; the latter require the signature $\sigma.n$ to be new. The semantic motivation is that in the case of $\Box\phi$, the formula ϕ holds in all worlds accessible from the world in which $\Box\phi$ is evaluated, but no new accessibility tuples should be introduced where not require. In the case of $\Diamond\phi$, the existence of a world accessible from the initial world is guaranteed, but no identity assumptions to previously introduced possible worlds should be made for similar reasons as in the case of the ($\exists$)-rule.

2.2.5 Model extraction

One very useful property of proofs via analytic tableaus is that for invalid inferences, there is a straightforward way to extract a counter model from an open branch. As mentioned above, a tableau constitutes an attempt to falsify the conclusion while satisfying the premises (if any are present), and each branch represents one possibility for these assumptions to hold. Thus a branch where no contradiction arises from these assumptions stands for a configuration in which the premises are true but the conclusion is false, in other words, a counter model.

Formally, the model extracted from an open branch $\mathcal{B}$ can be defined as follows: Let $\mathcal{B}$ be identified with the set of formulas occurring in it. Furthermore, let $\mathrm{prop}(\phi)$ denote the set of propositional variables occurring in ϕ, $\mathrm{const}(\phi)$ the set of constant symbols occurring in ϕ, and $\mathrm{pred}(\phi)$ the set of predicate symbols occurring in ϕ. We write $\mathcal{S}(\mathcal{B})$ for the model extracted from branch $\mathcal{B}$.

For propositional tableaus,

$$\mathcal{S}(\mathcal{B}) = \langle \mathcal{V} \rangle$$

with

$$\mathcal{V} : p \mapsto \begin{cases} \text{True} & \text{if } p \in \mathcal{B} \\ \text{False} & \text{if } \neg p \in \mathcal{B} \end{cases} \quad \text{for each } p \in \bigcup_{\phi \in \mathcal{B}} \mathrm{prop}(\phi).$$

That is, propositional variables occurring positively (that is, not negated) in the branch are interpreted as True, and propositional variables occurring negated in the branch are interpreted as False.

For first-order tableaus,

$$\mathcal{S}(\mathcal{B}) = \langle \mathcal{D}, \mathcal{I} \rangle$$

with

$$\mathcal{D} = \bigcup_{\phi \in \mathcal{B}} \mathrm{const}(\phi)$$

$$\mathcal{I} : P \mapsto \{\langle t_1, \ldots, t_n \rangle : P(t_1, \ldots, t_n) \in \mathcal{B}\} \quad \text{for each } P \in \bigcup_{\phi \in \mathcal{B}} \mathrm{pred}(\phi).$$

In other words, the domain of the structure consists of all constants occurring in the branch, and the interpretation function assigns to each predicate symbol those tuples of terms with which it occurs as a positive atomic proposition in the branch. Since constant symbols are interpreted as themselves, the obtained structure is a so-called term model.

For modal logics, the components will be relativized to possible worlds, corresponding to the signatures that the respective formula occurrences are annotated

with, and the set of worlds and the accessibility relation are extracted from the signatures in a straightforward manner.

Models extracted in this way are, in the general case, partial models in the sense that non-logical symbols occurring in the assumptions but not in the open branch are uninterpreted. However, these partial models can be extended to complete models by adding arbitrary interpretations to the otherwise uninterpreted symbols. In addition, it may be possible to add more tuples to existing interpretations and to add more objects to the domain while preserving satisfaction. Algorithms for so-called complete tableaus, which always yield complete models, can be given but are outside the scope of this work.

Taking up the open tableau (3) again, the countermodel extracted from the open branch ending in line 4 is $\mathcal{S}_2 = \langle \mathcal{V}_2 \rangle$ with

$$\begin{aligned} \mathcal{V}_2 : p &\mapsto \text{False} \\ q &\mapsto \text{True} \\ s &\mapsto \text{False} \end{aligned}$$

This countermodel is a partial model since the valuation for r is undefined, as r does not occur in the branch. An extension with $r \mapsto$ True or $r \mapsto$ False, respectively, yields two complete countermodels.

2.2.6 Tableau algorithm

For the purpose of implementing tableaus, it is preferable to use the procedure of so-called *systematic tableaus* as defined in (Smullyan, 1968, p. 59), which has the property that it always finds a closed tableau if one exists.

The complete algorithm used in our implementation for setting up a tableau, from the initial function call via the systematic tableau development to displaying the output, can be summarized as follows:

- set initial settings, add assumptions (`init`)
- while there are applicable rules: expand the tree (`expand`)
  - check whether tree expansion can be stopped
    - tree gets too big or
    - requested number of models found or
    - tree contains partial tree which is a validity tableau with no more applicable rules
  - recompute list of applicable rules
    - traverse all source nodes (= lines to potentially expand)
    - compile applicability entry
      - get applicable rules (rule type, rule name, formulas to add) from class belonging to formula in `expr.py`
      - select target nodes (= nodes to append the new lines to)
      - compute additional function arguments (such as list of constants occurring in the branch)
    - add rule of form (target, source, rule name, rule type, arguments, number of applications) to list of applicable

  - remove inactive branches from applicable
  - rank list according to specified criteria
- pick the topmost of the prioritized applicable rules
- apply the rule by calling the rule type function (`rule_alpha` etc.)
  - if appropriate, pick constant/world instantiation
  - add new node to target (`add_node`)
- check if branch is now closed, terminally open, or probably infinite
- if open, extract model (`model`)
  - collect propositional variables, constants, worlds and accessibility pairs on branch
  - collect literals (atoms and negated atoms) in the branch and turn into valuation/interpret
- print results (`show`)

2.3 Model generation with satisfiability tableaus

While model checking makes it possible to determine the truth value of a sentence in a given domain, this nevertheless presupposes that such a model is already defined. It would ease the analysis of potentially ambiguous expressions if models and countermodels to expressions could by found by the computer itself. That is, we are looking for way of performing *model generation*.

The tableau method as presented above could now be used not only to prove inferences as valid, but also to find models for given (sets of) sentences, by starting the tree with the positive sentence ϕ (instead of $\neg\phi$) and expanding the tableau until an open branch is found, from which a model satisfying the assumptions can then be extracted.

Tableaus with the conventional rule formulation ("validity tableaus") are not in general suited for this purpose, unfortunately. The following two kinds of problems arise.

First, models obtained with validity tableaus are not always minimal, in the sense of containing a miniimal number of objects in the domain of discourse. This is due to the constraint that every existential formula ought to be instantiated with a new constant, thereby introducing as many individuals as existential claims occur in the assumptions, although it may be possible to satisfy several existential propositions with the same witness.

While minimality is often a desired property in mathematical problems, one may wonder why this is relevant in linguistic applications. Typically, entities introduced by quantifiers such as the determiner *a* are usually intended to be distinct; for instance, in a sentence like

(4) A student reads a book. one would not want to identify the student with the book. There are, however, cases where identifying two existentially quantified objects is clearly desirable, in particular, in discourses with repeated existential quantification across sentence boundaries.

(5) a. There are two birds in the garden.
    b. A blackbird is singing and a sparrow is taking a bath.

c. Blackbirds are not sparrows.

The first two sentences could naturally occur in a discourse; the last sentence is a background assumption based on world knowledge.

This discourse can be formalized as

(6) a. $\exists x \exists y(x \neq y \wedge \mathit{Bird}(x) \wedge \mathit{Bird}(y) \wedge \mathit{Garden}(x) \wedge \mathit{Garden}(y))$
b. $\exists x(\mathit{Sparrow}(x) \wedge \mathit{Chirp}(x)) \wedge \exists y(\mathit{Blackbird}(y) \wedge \mathit{Bath}(y))$
c. $\forall x \neg(\mathit{Sparrow}(x) \wedge \mathit{Blackbird}(x))$

The conventional tableau rules would result in the four existential quantifications in (6-a) and (6-b) to be instantiated with four different individuals, but it is more plausible to assume that the blackbird and the sparrow are identical to the two birds introduced in the first sentence.

Second, and even more critical, is the fact that there are satisfiable sentences for which the conventional tableau algorithm does not yield any model at all. For instance, consider the reverse of the above discussed quantifier commutativity:

$$\forall x \exists y \mathit{Loves}(x, y) \not\models \exists y \forall x \mathit{Loves}(x, y) \tag{1}$$

Since this inference is invalid, it should be possible to find a model for the set of sentences $\{\forall x \exists y \mathit{Loves}(x, y), \neg \exists y \forall x \mathit{Loves}(x, y)\}$, that is, a structure in which everyone heard someone but there is no unique person that everyone heard. However, applying validity tableaus to these assumptions leads to an infinite branch that neither closes nor can be terminated to constitute an open branch:

1.	$\neg \exists y \forall x \mathit{Loves}(x, y)$	(A)
2.	$\forall x \exists y \mathit{Loves}(x, y)$	(A)
3.	$\neg \forall x \mathit{Loves}(x, a)$	$(\neg\exists,\ 1,\ [y/a]*)$
4.	$\exists y \mathit{Loves}(a, y)$	$(\forall,\ 2,\ [x/a])$
5.	$\neg \mathit{Loves}(b, a)$	$(\neg\forall,\ 3,\ [x/b]*)$
6.	$\mathit{Loves}(a, c)$	$(\exists,\ 4,\ [y/c]*)$
7.	$\neg \forall x \mathit{Loves}(x, b)$	$(\neg\exists,\ 1,\ [y/b])$
8.	$\neg \mathit{Loves}(d, b)$	$(\neg\forall,\ 7,\ [x/d]*)$
9.	$\exists y \mathit{Loves}(b, y)$	$(\forall,\ 2,\ [x/b])$
10.	$\mathit{Loves}(b, e)$	$(\exists,\ 9,\ [y/e]*)$
	$\vdots$	

The problem is that with the conventional method, existential formulas need to be re-evaluated with a new constant as long as no contradiction has been found. In the present example, no such existential instantiation can possibly yield a contradiction, and therefore the tableau algorithm never terminates. Similar problems arise for modal sentences.

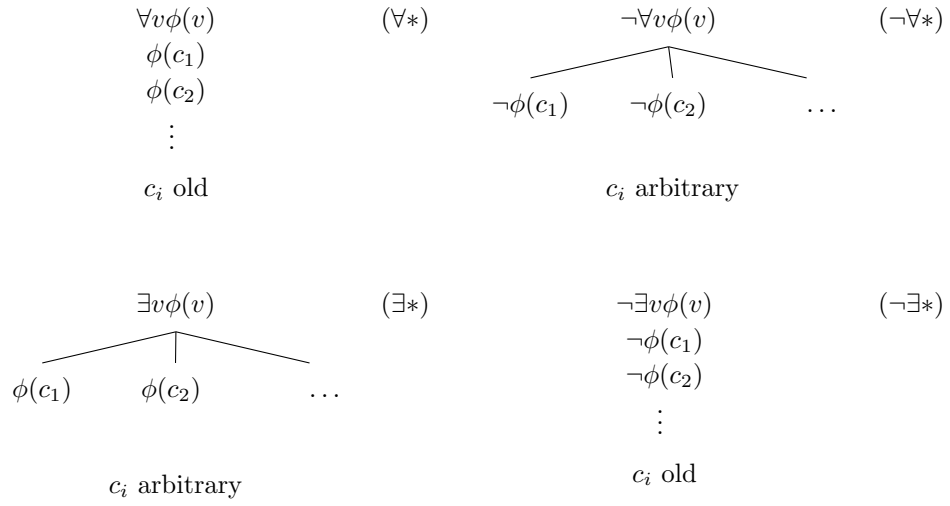
To mitigate this situation, we consider a modification of the validity tableaus, originally formulated for first-order logic by Boolos (1984). This gives rise to a new variant of tableaus, called *satisfiability tableaus* in the following.

The modification is twofold: First, since we now are interested in finding models rather than closed tableaus, the algorithm will try to produce open branches and terminate once the desired number of them has been found, rather than trying to re-evaluate formulas until a contradiction has been found. Second, rules for quantifiers

and modal operators will be modified so as to yield models with a domain as small as possible.

2.3.1 Satisfiability tableaus for first-order logic

The modified rules for quantifiers are as follows:



The major modification concerns the rule for the existential quantifier: Existential formulas may be instantiated with arbitrary constants, even ones that have already been introduced in the branch. The algorithm will first try to identify the existentially quantified individual with a previously introduced constant in order to preserve minimality. If this fails, the rule branches into alternative instantiations, until a suitable model has been found. Universal formulas can and must be instantiated with all constants occurring in the branch in order to judge a branch open.

Satisfiability tableaus are complete for finite satisfiability, meaning that for arbitrary sets of premises a finite model will be found if one exists (Boolos, 1984).

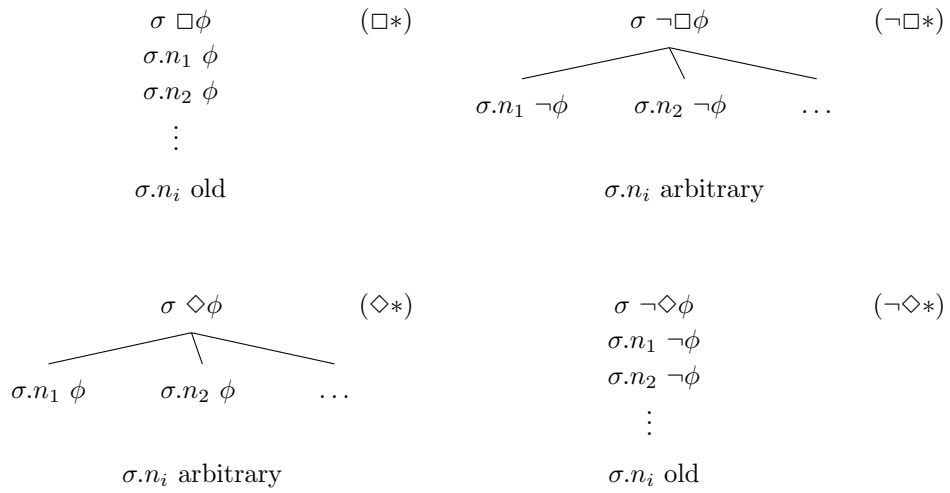
Reconsidering the above example 5, applying model generation in pyPL using these satisfiability tableau rules correctly produces a model $\mathcal{S}_2 = \langle \mathcal{D}_2, \mathcal{I}_2 \rangle$ with

$$
\begin{array}{lll}
\mathcal{D}_2 = \{\mathrm{a}, \mathrm{b}\} & & \\
\mathcal{I}_2: & \mathit{Bath} & \mapsto \{\langle \mathrm{a} \rangle\} \\
& \mathit{Bird} & \mapsto \{\langle \mathrm{a} \rangle, \langle \mathrm{b} \rangle\} \\
& \mathit{Blackbird} & \mapsto \{\langle \mathrm{a} \rangle\} \\
& \mathit{Chirp} & \mapsto \{\langle \mathrm{b} \rangle\} \\
& \mathit{Garden} & \mapsto \{\langle \mathrm{a} \rangle, \langle \mathrm{b} \rangle\} \\
& \mathit{Sparrow} & \mapsto \{\langle \mathrm{b} \rangle\}
\end{array}
$$

In this model, the blackbird and the sparrow are identified with the birds introduced in the first sentence as desired, but not with each other, as the axiom provided as the third premise rules this out.

2.3.2 Satisfiability tableaus for modal logic

In a similar manner, rules for modal operators can be modified to make tableaus more suitable for model generation:



The structure of the rules is analogous to those for quantifiers: Formulas of the form $\Diamond\phi$ are evaluated by branching between alternatives of possible worlds with a preference to avoid the introduction of new worlds and new accessibility claims. Formulas of the form $\Box\phi$ have to be evaluated in every world introduced as accessible in order to ensure that all assumptions are satisfied in the branch.

An example of how these rules can be put to use for modal model generation is discussed in Section 3.2.

2.4 Limitations

While the methods discussed here turn out fruitful for the treatment of simpler textbook examples used in education, the practical use suffers from severe limitations due to the complexity of real-life problems on the one hand, and theoretical constraints lying in the nature of first-order logic on the other hand.

2.4.1 Complexity

One particularly illustrative example of where pyPL in its current state of development runs into problems is the following scenario:

(7) a. Every tupperware box has a fitting lid.
b. There is no lid that fits on all tupperware boxes.
c. Tupperware boxes are not lids.
d. Tupperware boxes exist.

To the human reader it is immediately obvious how this discourse can be satisfied: Required is a domain with four elements, two of them being tupperware boxes and the other two lids, and each of the lids fits on one of the boxes. For a naive computer program, however, this is by far not trivial. The combinatorial possible interpretations

for the given set of non-logical symbols on a domain with four elements make for 2^{24} possible structures, out of which only 2^5 – which roughly amounts to every 100.000'th branch – are models of the theory. With the current implementation, this degree of complexity is practically intractable: When running pyPL on the input, even after a whole minute a mere few thousand branches have been explored and no model found.

The relative simplicity of this already problematic example clearly shows that the naive implementation chosen by us is completely unsuitable for real-life problems with much larger input. Although some heuristics can be and have been implemented to make the proof search procedure more efficient, tableau trees are inherently vulnerable to combinatorial explosion. If performance, rather than theoretical insight is a goal, an altogether different proof system may be more suitable.

In addition to the theoretical foundation chosen, the choice of Python as the programming language makes many aspects of implementation straightforward and is especially suited for didactic purposes, it is inferior to many other programming languages in terms of execution time. Nevertheless, the current implementation leaves room for several efficiency improvements, in particular concerning the search for and prioritization of applicable rules.

2.4.2 Completeness and decidability

A further problem is that of decidability: The tableau calculus is complete for first-order validity and will eventually detect all valid inferences as such. But first-order logic is not co-semi-decidable, that is, it is impossible to find an algorithm that detects all invalid inferences as such (Church, 1936; Turing, 1936). However, given the completeness results of the tableau calculi, the combination of validity tableaus and satisfiability tableaus makes it in principle possible to detect all valid inferences and all invalid inferences with finite countermodels. The only problems which are outside the theoretical scope of the combined tableau approach are invalid inferences which only have infinite countermodels or, respectively, sets of sentences that can only be satisfied by infinite models. The problem of complexity seems the practically more significant one.

3 Linguistic phenomena

After this discussion of the theoretical basis, we now consider a selection of linguistic phenomena with respect to their treatability in formal logic, and in particular in the pyPL system.

The core idea of formal natural language semantics is to translate natural language into an intermediate formal language – for instance, first-order logic – for which a more precisely definable and computable semantics can be defined. Under the assumption that the formal translation is adequate, and that the formal logical rules match speaker's intuitions of reasoning, this formal semantics indirectly gives a natural language semantics.

3.1 Ambiguity

One crucial feature of natural language that sets it apart from formal languages is ambiguity, that is, the possibility for one expression to have more than one meaning.

In order to automatically find different readings of a linguistic expressions, one requires a parser that will convert a natural language expression into a formal representation. This is a difficult task, and pyPL does not contain such a natural language parser.

Nevertheless, the system can be used to aid the detection of of ambiguity: If two sentences are to differ in meaning, there must be at least one situation in which one is true while the other is false. This is known as Cresswell's most certain principle (Zimmermann & Sternefeld, 2013, p. 28):

> If a (declarative) sentence S_1 is true and another sentence S_2 is false in the same circumstances, then S_1 and S_2 differ in meaning.

Hence, a test for ambiguity can be carried out by finding a structure which is a model of one sentence but not of the other.

There are three different kinds of ambiguity to distinguish, which will be discussed in the following sections.

3.1.1 Lexical ambiguity

Lexical ambiguity is ambiguity arising due to a word having more than one meaning. An example is the following (cf. Kroeger, 2019, p. 23):

(8) I just turned 51, but I have a nice new organ which I enjoy tremendously.

The word *organ* is lexically ambiguous: It could mean a musical instrument, or a body organ.

It is commonly assumed that the two meanings of *organ* belong to two different lexical entries, which only happen to share the same phonetic form but have different semantic values. When translating such expressions into a formal language, one would assume two different non-logical symbols for the two lexical entries, thereby resolving the ambiguity on a lexical level. This becomes more apparent when a lexically ambiguous word is used in two meanings within the same sentence, such as

(9) I've always been hard of hearing, but playing my organ on full volume further impaired my organ.

An adequate formalization of this sentence demands the use of two distinct predicate symbols, *organ_mus* and *organ_phys*, that is, a distinction has to be made on the formal lexical level. Since formal interpretations are functions, in order for two different objects to end up as the denotations of *my organ*, two different functions belonging to two different lexical items have to be assumed.

With an appropriate sentence formalization and structure definition, model checking reveals that the two occurrences of the linguistic expression *organ* can indeed evaluate to two different objects in the same model.

3.1.2 Structural ambiguity

Structural ambiguity arises when a phrase has two different syntactic parses, while each individual word is unambiguous.

A frequently cited example is (cf. Zimmermann & Sternefeld, 2013, p. 26):

(10) John saw the man with the binoculars.

This sentence has two readings, according to two different syntactic structures:

a. John used binoculars to observe a man
   [John [[saw [the man]]] with the binoculars]
b. John observed a man who had binoculars with him
   John [saw [[the man] with the binoculars]]

The difference in syntactic structure is the location of attachment for the propositional phrase: In reading (a), *with the binoculars* is an adjunct of the verb phrase *saw the man*, thereby serving as an instrumental modifier to the seeing event, whereas in (b), *with the binoculars* is adjoined to the determiner phrase *the man*, thereby giving further information on the object.

Another source of structural ambiguity is coordination:

(11) Mary is happy and Susan is happy or Lea is happy.

The two readings of this sentence obtained by the two different parses are:

a. [Mary is happy] and [Susan is happy or Lea is happy]
b. [Mary is happy and Susan is happy] or [Lea is happy]

If the two sentences are to differ in meaning, then according to Cresswell's principle there should be a structure in which one sentence is true and the other is false. Such a structure is $\mathcal{S}_1 = \langle \mathcal{D}_1, \mathcal{I}_1 \rangle$ from above, for which we repeat the relevant parts here:

$$\begin{aligned} \mathcal{D}_1 &= \{\text{John}, \text{Lea}, \text{Mary}, \text{Peter}, \text{Susan}\} \\ \mathcal{I}_1 : \quad & \textit{lea} \quad \mapsto \text{Lea} \\ & \textit{mary} \quad \mapsto \text{Mary} \\ & \textit{susan} \quad \mapsto \text{Susan} \\ & \textit{Happy} \mapsto \{\langle \text{John} \rangle, \langle \text{Lea} \rangle, \langle \text{Peter} \rangle, \langle \text{Susan} \rangle\} \end{aligned}$$

A formalization of the two sentences is

(12) a. $\textit{Happy}(\textit{mary}) \wedge (\textit{Happy}(\textit{susan}) \vee \textit{Happy}(\textit{lea}))$
     b. $(\textit{Happy}(\textit{mary}) \wedge (\textit{Happy}(\textit{susan})) \vee \textit{Happy}(\textit{lea})$

Running model generation in pyPL on the negation of (12-a) and (12-b) will produce the partial model $\mathcal{S}_8 = \langle \mathcal{D}_8, \mathcal{I}_8 \rangle$ with

$$\begin{aligned} \mathcal{D}_8 &= \{\text{lea}, \text{mary}, \text{susan}\} \\ \mathcal{I}_8 : \quad & \textit{Happy} \mapsto \{\langle \text{lea} \rangle\} \end{aligned}$$

which satisfies (11-b) but not (11-b), which proves their difference in meaning.

3.1.3 Scopal ambiguity

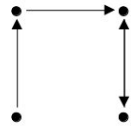
Scopal ambiguity is the phenomenon of one lexical and syntactic structure corresponding to more than one logical form.

Consider the following sentence (cf. Chierchia, 1990, p. 32):

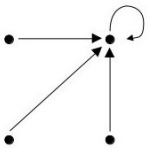
(13) Everyone loves someone.

This sentence has two readings:

a. For everyone there is someone they love, but different people may love different persons.
Formally: $\forall x \exists y Loves(x, y)$.
Diagrammatically:



b. There is a common person who everyone loves.
Formally: $\exists y \forall x Loves(x, y)$.
Diagrammatically:



Reading (b) is called the linear reading, because the logical form, in particular, the scopal relationships between the quantifiers $\forall$ and $\exists$, is in line with the surface structure. Reading (a) is called the inverted reading, because the scope is reversed with respect to the order in which the quantifiers appear in the natural language sentence. It is important to note that neither the lexical items *everyone* and *someone* are ambiguous, nor are there two different parses of the sentence structure; unambiguously *someone* is the subject and *everyone* the object. It is only on the abstract level of logical form that the two different meanings come about.

An interesting question is whether the two readings are independent or entail each other. That is, is every situation in which (13-a) is true also a situation in which (13-b) is true and vice versa? This is a matter of logical inference, and the problem can be resolved by theorem proving on first-order formulas. Regarding our example, we have that

$$\exists y \forall x Loves(x, y) \vDash \forall x \exists y Loves(x, y)$$
$$\forall x \exists y Loves(x, y) \nvDash \exists y \forall x Loves(x, y)$$

A semantic tableau will reveal that the inference from reading 2 to 1 is valid, whereas the reverse direction is not, and will compute a countermodel for the latter:

$$\begin{aligned} &\mathcal{S}_2 = \langle \mathcal{D}_2, \mathcal{I}_2 \rangle \text{ with} \\ &\mathcal{D}_2 = \{\mathrm{a}, \mathrm{b}\} \\ &\mathcal{I}_2 : \quad Loves \mapsto \{\langle \mathrm{a}, \mathrm{b} \rangle, \langle \mathrm{b}, \mathrm{a} \rangle\} \end{aligned}$$

This structure represents a situation in which for everyone there is someone they love (namely, a loves b and b loves a), but there is no one person who loves everyone.

Again, differences in formal satisfiability reveal a difference in linguistic meaning.

3.2 Modality

Of further interest in linguistics are modal expressions such as *must* and *might*. It is easy to see that these operators are not truth-functional, hence a formal treatment requires an extension of the language of standard first-order logic. The field of modal

logic is vast and there are many different modalities to consider. For the purpose of this discussion we restrict ourselves to the alethic modalities of possibility and necessity.

3.2.1 Constant vs. varying domains

Consider the following example:

(14) A comet might hit and destroy earth.

This sentence contains the modal operator *might*, commonly symbolized as $\Diamond$. The sentence can be paraphrased in two different ways:

a. There exists a comet which in some hypothetical situation is going to hit earth.
   Formally: $\exists x \Diamond \mathit{Comet}(x)$
b. There is a hypothetical situation in which a comet is going to hit earth.
   Formally: $\Diamond \exists x \mathit{Comet}(x)$

Intuitively, (14-2) is the desired interpretation, whereas the wording in (14-1) seems too strong. The difference between the formalizations (14-1) and (14-2) is once again one of scope: Under the linear reading $\exists x \Diamond \mathit{Comet}(x)$, the existential quantifier takes scope over the modal operator, whereas in the inverted reading $\Diamond \exists x \mathit{Comet}(x)$, *might* semantically scopes over *a*. The question is whether these are logically equivalent paraphrases of the original sentence or different readings corresponding to different meanings.

A semantics for modal logic is commonly given in terms of so-called Kripke structures. A Kripke structure contains a set of so-called possible worlds $\mathcal{W}$, and a binary relation $\mathcal{R}$ on $\mathcal{W}$, called accessibility relation. Intuitively, for the purpose of alethic modalities, a possible world is as a hypothetical or actual situation, and a world w accessing another world w' can be thought of as the state of affairs w' being imaginable from the perspective of w. Truth is now relative to worlds, that is, modal interpretations are functions from non-logical symbols to functions from possible worlds into objects and relations over the domain.

A sentence of the form *It is possible that* ϕ is true at a world w iff there is at least one world w' accessible from w in which ϕ holds, and *It is necessary that* ϕ is true at w iff there is no world w' among the worlds accessible from w at which ϕ is false. Formally (cf. Gamut, 1991, p. 23),

$$[\![\Diamond\phi]\!]^{\mathcal{S},v,w} = \text{True} \iff \text{there exists } w' \in \mathcal{W}\text{: } w\mathcal{R}w' \text{ and } [\![\phi]\!]^{\mathcal{S},v,w'} = \text{True};$$
$$[\![\Box\phi]\!]^{\mathcal{S},v,w} = \text{True} \iff \text{for all } w' \in \mathcal{W}\text{: if } w\mathcal{R}w' \text{ then } [\![\phi]\!]^{\mathcal{S},v,w'} = \text{True}.$$

Coming back to the original example, the question whether the two paraphrases of (14) are equivalent, that is, whether

$$\exists x \Diamond \mathit{Comet}(x) \dashv\models \Diamond \exists x \mathit{Comet}(x)$$

holds, depends on whether the modal first-order logic is assumed to have so-called constant domains or varying domains. With constant domains, all possible worlds in a structure share the same domain of discourse, whereas with varying domains, each possible world is assigned its own domain of individuals. From an epistemic point

of view, the assumption that different objects may exist in different possible states of affairs seems plausible; hence a modal logic with varying domains may be more appropriate for the situation at hand.

Sound and complete tableau rules can be given for each of these two semantics. Running the theorem proving mode in pyPL on the two formulas, configuring the program to use tableaus for modal logic with constant domains will produce a proof that the two sentences in our example are equivalent, whereas using tableaus for varying domains generates a countermodel $\mathcal{S}_3 = \langle \mathcal{W}_3, \mathcal{R}_3, \mathcal{D}_3, \mathcal{I}_3 \rangle$, where

$$\begin{array}{lll} \mathcal{W}_3 = & \{w_1, w_2\} & \\ \mathcal{R}_3 = & \{\langle w_1, w_1\rangle, \langle w_1, w_2\rangle\} & \\ \mathcal{D}_3 : & w_1 \mapsto \{\mathrm{a}\} & \\ & w_2 \mapsto \{\mathrm{b}\} & \\ \mathcal{I}_3 : & C \mapsto w_1 \mapsto \{\} & \\ & \qquad\quad w_2 \mapsto \{\langle \mathrm{b}\rangle\} & \end{array}$$

This structure formalizes a situation in which there is a possible world w_2 accessible from w_1 where there exists an object that is a comet, but no such comet exists in the actual world w_1. Again, the formal computation adequately reflects the intuitive understanding of the linguistic meaning of (14) in the sense of (b).

3.2.2 Other modal frames

If the two modal operators introduced above are to be interpreted as alethic modalities, i.e., $\Diamond\phi$ is taken to mean *It is possible that* ϕ and $\Box\phi$ as *It is necessary that* ϕ, then the rules of modal logic assumed by default are not sufficient to account for the intuitions most speakers have about these expressions in natural language. For instance, from *It is necessary that* ϕ one would want to infer that *It is possible that* ϕ and also that *It is the case that* ϕ. However, neither of the inference $\Box\phi \vDash \Diamond\phi$ nor $\Box\phi \vDash \phi$ are valid in the standard modal logic K which is used in pyPL.

In order to derive these inferences, additional axioms (from a rule-based perspective) or constraints on the accessibility relation (from a model-theoretic perspective) are required. The inference from necessity to possibility is given in the modal system D, which can be obtained by adding to K the axiom $\Box p \to \Diamond p$ (Fitting & Mendelsohn, 1998, p. 71), or equivalently, defining the accessibility relation on the possible worlds to be serial. (This is the case iff for every $w \in W$ there exists some $w' \in W$ such that wRw'; in other words, there are no "dead ends"). The inference from necessity to actuality can be arrived at by adding the rule $\Box p \to p$, or alternatively, defining the accessibility relation to be reflexive, yielding system T (ibid.). If the accessibility relation is interpreted as something similar a world being able to imagine another, then it seems plausible to assume that every world can access its own propositions. Another modal system commonly employed is $\mathsf{S5}$, in which all modal truths are necessary.

For each axiom and accessibility characterization a corresponding set of tableau rules can be given, so that tableau calculus extended with these rules is sound and complete for the respective modal logic (Fitting & Mendelsohn, 1998, p. 52). pyPL does not currently implement these rules, hence the inferences above are not derivable in the system. An extension is, however, straightforward and is a possibility for future development.

3.3 Tense and aspect

Every natural language is, in one way other the other, able to express tense – past, present and future – , often in combination with aspect, such as progressive, perfect or habitual Kearns (2011). A formal semantics should be able to syntactically represent and adequately interpret tense and aspect.

Tense logic is a logic dedicated to expressing tense and is very similar to and can be combined with modal logic. Analogous to the set of possible worlds and an binary accessibility relation on them, models of tense logic contain a set of points in time and a binary relation on them which is interpreted as expressing the earlier-than relation. Tense operators such as (Gamut, 1991, p. 23)

(15) a. Pϕ: it was at some state in the past the case that ϕ
b. Gϕ: it is always going to be the case that ϕ can then be defined by quantifying over moments in time.

Model-theoretic evaluation rules and tableau rules can be given for this kind of logic, but are not at the moment implemented in pyPL, so tense and aspect can not systematically be accounted for in the system.

3.4 Intensional contexts

Related to modality is the concept of intensionality. Consider the following example from (Zimmermann & Sternefeld, 2013, p. 174):

(16) a. John knows that Hamburg is larger than Cologne.
b. John knows that Pfäffingen is larger than Breitenholz.

The attitude verb *knows* here is a verb which takes as arguments a determine phrase and a sentence. Under a naive approach, truth conditions for this kind of verbs operate on the arguments' extensions, that is, a pair of an individual and a truth value:

$$[\![Know(a, s)]\!]^{\mathcal{S},v} = \text{True} \iff \langle [\![a]\!]^{\mathcal{S},v}, [\![s]\!]^{\mathcal{S},v} \rangle \in [\![Know]\!]^{\mathcal{S},v}.$$

This account is inadequate: Since the denotations of sentences are truth values, all sentences that are true in a given situation will have the same extension, and the same goes for all false sentences. Hence, as soon as an individual knows one true proposition, there is the pair $\langle \text{John}, \text{True} \rangle$ in the interpretation of an attitude verb, and with it every other true sentences; analogous for false sentences. However, just because John knows that Hamburg is larger than Cologne does not mean that he also knows that Pfäffingen is larger than Breitenholz. Truth values thus does not suffice to properly formalize the semantics of attitude verbs.

Another problem is exemplified by the following scenario (adapted from Quine, 1956, p. 178):

(17) a. Joe Biden is a democrat.
b. Donald Trump is not a democrat.
c. Joe Biden is the president elect of the U.S.
d. Mary believes that Donald Trump is the president elect of the U.S.
e. Mary believes that Joe Biden is a democrat.
f. Mary does not believe that the president elect is a democrat.

Under a naive approach, the sentences would be formalized as follows, with *believe* acting as a predicate on pairs of individual terms and sentences:

(18) a. $Democrat(biden)$
b. $\neg Democrat(trump)$
c. $biden = president$
d. $Believe(mary, trump = president)$
e. $Believe(mary, Democrat(biden))$
f. $\neg Believe(mary, Democrat(president))$

However, this leads to an inconsistency. Since the tuple

$$\langle [\![mary]\!]^{\mathcal{S},v}, [\![trump = president]\!]^{\mathcal{S},v} \rangle$$

is in the interpretation of *Believe*, then with

$$\langle [\![mary]\!]^{\mathcal{S},v}, [\![Democrat(biden)]\!]^{\mathcal{S},v} \rangle \in \mathcal{I}(Believe)$$

and

$$biden = president,$$

by substitution it so also be the case that

$$\langle [\![mary]\!]^{\mathcal{S},v}, [\![Democrat(president)]\!]^{\mathcal{S},v} \rangle \in \mathcal{I}(Believe),$$

but this contradicts the premise that

$$\neg Believe(mary, Democrat(president)).$$

Nevertheless, the assumptions are intuitively consistent: Mary may very well hold beliefs that contradict reality, as long as the facts deemed actually true are consistent. It may even be legitimate to assume that an individual's set of beliefs is inconsistent, given that humans are not perfect rational beings.

A way out of this is, again, Kripke semantics, where possible worlds serve to reflect a speaker's epistemic or doxastic alternatives. A distinction is made between the propositions that hold true in the actual world and those which are known or believed by an individual, and sentences may have different extensions in the same structure depending on the possible world. For instance, in the situation described above, the critical difference lies in the denotations of the referential expressions *biden* and *president* between the real world and Mary's beliefs:

expression	denotation in real world	denotation in Mary's world
biden	Joe Biden	Joe Biden
$Democrat(biden)$	True	True
president	Joe Biden	Donald Trump
$Democrat(president)$	True	False

Since $biden = president$ is true in the real world but not in Mary's world, the above substitution argument does not apply, and the *believe*-claims entail no contradiction to the facts judged as true in the real world.

This argument shows that the predicates *know* and *believe* are intensional contexts, that is, expressions operating on intensions, rather than on extensions. Formally, this

can be dealt with by introducing an intension operator $^\wedge$ alongside an extension operator $^\vee$ (cf. Gamut, 1991, p. 123):

$$[\![^\wedge\alpha]\!]^{\mathcal{S},v,w} = h : W \to \mathcal{D}_\alpha : w' \mapsto [\![\alpha]\!]^{\mathcal{S},v,w'};$$
$$[\![^\vee\alpha]\!]^{\mathcal{S},v,w} = [\![\alpha]\!]^{\mathcal{S},v,w}(w).$$

The intension operator $^\wedge$ computes the intension of its argument expression, whereas the extension operator $^\vee$ reverses the intensionalization and returns the extension at the given world.

The interpretation of attitude verbs is then a relation between individuals and intensions of sentences, that is, instead of the extensional account as tentatively developed above, we have the following truth condition:

$$[\![Know(a,s)]\!]^{\mathcal{S},v} = \text{True} \iff \langle [\![a]\!]^{\mathcal{S},v}, [\![^\wedge s]\!]^{\mathcal{S},v}\rangle \in [\![Know]\!]^{\mathcal{S},v}.$$

This solves both of the above problems. In (16), the intensions between the sentences *Hamburg is larger than Cologne* and *Pfäffingen is larger than Breitenholz* differ, since is possible to imagine a world where one is true and the other is false. Therefore, the intension of one sentence being in the interpretation of *Know* does not entail that the other one is, too. Hence, the semantics now adequately reflects that John may know one fact but not the other, although both are true in the actual world. In (17), rather than the formalizations proposed in (18), we have

(19)
a. $Democrat(biden)$
b. $\neg Democrat(trump)$
c. $biden = president$
d. $Believe(mary, {}^\wedge trump = president)$
e. $Believe(mary, {}^\wedge Democrat(biden))$
f. $\neg Believe(mary, {}^\wedge Democrat(president))$

A modal first-order structure with an interpretation for the *Believe* predicate can be defined accordingly, that is, containing tuples of individuals and functions from possible worlds to truth values. Model checking in pyPL on such a structure verifies that it is indeed possible for all of the above sentences to be true in the same structure and possible world.

3.5 Generalized quantifiers

So far, we have worked with first-order logic, with modal extensions added. However, this formal language is not expressive enough to account for all constructions that occur in natural language, an example being generalized quantifiers (cf. Westerståhl, 2019, §4, Definition 1). In particular, it can be shown that the quantifiers *most* and *more than*, as in

(20)
a. Most women are happy.
b. More women than men are happy.

are not first-order definable (Peters & Westerståhl, 2006, p. 468). However, it is possible to define these quantifiers set-theoretically, namely as a relation between sets, with truth conditions defined by comparing the sets' cardinalities:

$$[\![most\ x(\phi,\psi)]\!]^{\mathcal{S},v,w} = \text{True} \iff$$

$$|\{a : [\![\phi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\} \cap \{a : [\![\psi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\}| >$$
$$|\{a : [\![\phi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\} - \{a : [\![\psi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\}|$$

$$[\![\textit{more } x(\phi,\chi,\psi)]\!]^{\mathcal{S},v,w} = \text{True} \iff$$
$$|\{a : [\![\phi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\} \cap \{a : [\![\psi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\}| >$$
$$|\{a : [\![\chi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\} \cap \{a : [\![\psi]\!]^{\mathcal{S},v^{[x\mapsto a]},w} = \text{True}\}|$$

Model checking for the two quantifiers *most* and *more than* is supported in pyPL by implementing these set-theoretic definitions. Systematic reasoning about these expressions by defining sound and complete tableau rules remains an interesting problem to be explored in future work.

3.6 Donkey sentences, anaphora and discourse

So-called donkey sentences have becomes famous in linguistics for posing problems for classical predicate logic and Montague semantics:

(21) Every farmer who owns a donkey pets it.

This sentence (adapted from Gamut, 1991, p. 270) contains an existential quantifier in a relative clause modifying a universal sentence, and a pronoun in the verb phrase interpreted as anaphorically referring to the existentially introduced individual. However, formalizing this sentence in predicate logic is difficult: Representing the sentence as

(22) $\forall x((\mathit{Farmer}(x) \wedge \exists y \mathit{Donkey}(y)) \rightarrow \mathit{Pet}(x,y))$

as the syntactic structure would suggest is inadequate, because the occurrence of variable y in the succedent of the implication is not, as intended, in the scope of the existential quantifier. A formalization that adequately captures the sentence's truth conditions is

(23) $\forall x \forall y((\mathit{Farmer}(x) \wedge \mathit{Donkey}(y)) \rightarrow \mathit{Pet}(x,y))$

but the problem is how to systematically arrive at this representation from the natural language sentence. It is unclear how a natural language *a* in adjunct position could end up as a universal quantifier scoping over the entire sentence, and a syntactic transformation is not possible, because as far as logical equivalence is concerned, moving a quantifier from inside the antecedent of an implication to outside of the implication and changing to its dual is only permitted if this does not result in the binding of variables in the succedent – but binding the anaphoric pronoun y in the verb phrase is precisely what is desired here.

A similar problem arises for discourses, that is, sequences of sentences, that involve quantification: In

(24) A man walks in the park. He whistles.

(Gamut, 1991, p. 267), the pronoun *he* should be bound by the *a* quantifier, but if each sentence is treated as a formula and sentences in sequence are treated as conjoined, then the variable in the second sentence incorrectly ends up outisde the scope of the existential quantifier in the first sentence:

(25) $\exists x(Man(x) \wedge Walk(x)) \wedge Whistle(x)$

The truth conditions can be correctly captured by letting the existential quantifier scope over the entire conjunction, but it is again questionable how to systematically arrive at this formalization, because obviously not every quantifier occurrence can arbitrarily be re-scoped.

Two proposed solutions for this are *Discourse Representation Theory (DRT)* and *Dynamic Predicate Logic (DPL)*. Discourse representations theory introduces intermediate representations, called discourse representation structures (DRS's), intended to store information conveyed by multi-sentence discourses in structures involving nested levels of boxes. Dynamic predicate logic is one way to define a semantics for DRS's, which relies on dynamic modification of variable assignments.

Implementations of DRT and DPL have been done, see, for instance as presented in Blackburn & Bos (1999). They are is, however, outside the scope of pyPL's range of functionality, so trans-sentential variable binding is another class of problem not analyzable with the method of standard model-theoretic evaluation and semantic tableaus.

4 Outlook

The system in its current state lends itself to a vast number of possible extensions.

- Lambda calculus, higher-order logic and type theory: Simply typed lambda calculus with e and t as atomic types is de facto the lingua franca of formal natural language semantics. Adding support for function abstraction, expressions of higher order (such as expressions denoting sets of sets) and type annotation would greatly extend the expressive power of the formal language and thereby make it possible to treat a larger number of additional linguistic phenomena.

- More logics: Other modal frames such as `S5` and tense logic are natural and linguistically useful candidates for extensions of the current modal logic coverage. Fuzzy and probabilistic logic may be worthwile alternatives to the classical two-valued semantics.

- Other frameworks and calculi: The classical model checking could be linked up with other semantic frameworks, such as Discourse Representation Theory (as e.g. presented in Kamp & Reyle (1993)) and Dynamic Predicate Logic (Dekker (1993)) for better coverage of cross-sentential semantics. The provided proof methods could be extended to calculi other than analytic tableaus, such as natural deduction (see e.g. Prawitz (1965)).

- More world knowledge: Maintaining a database of axioms with world knowledge such as *Students are not books* would improve the quality of the models generated and perhaps make pyPL more useful for actual problem solving rather than just experimental exploration.

  Several possibilities for didactic applications come to mind:

  - Students can use the program to verify simple satisfactions and inferences, thereby getting more acquainted with the formal methods.

– For those who have been introduced to basic programming in Python, relating theory to implementation by investigating the code and stepwise tracing the computation process from function call to function call may help improve their understanding of both logic and programming techniques. Letting participants complete pieces of code in a cloze-like exercise setting is also a possibility.

– At a more advanced stage of linguistic education, nascent researchers may test their own theoretical proposals by defining appropriate clauses on top of an existing basic framework for formal reasoning.

5 Conclusion

We showed how computational semantic evaluation and a tableau-based implementation of first-order logic can be used to analyze a number of linguistic phenomena. We conclude that while naive methods can successfully model the basic ideas taught in elementary linguistics courses, the system in its current state is not suitable for more demanding academic or industrial applications. Further development could extend coverage as well as efficiency, and it is intended to make use of the tool in academic linguistic education.

References

Patrick Blackburn and Johan Bos, *Representation and Inference for Natural Language, Volume II: Working with Discourse Representation Structures*, University of Saarland, 1999.

George S. Boolos, Trees and finite satisfiability, *Notre Dame Journal of Formal Logic* 25 (1984), 110–115.

Gennaro Chierchia and Sally McConnell-Ginet, *Meaning and Grammar: An Introduction to Semantics*, MIT Press, 2000.

Alonzo Church, A Note on the Entscheidungsproblem, *The Journal of Symbolic Logic* **1** (1936), 40–41.

Paul J. E- Dekker, Transsentential meditations: Ups and downs in dynamic semantics, *Linguistics* 14 (1993), 65-117.

Johannes Dellert, *Non-minimal Model Generation for Natural Language Semantics*, unpublished, 2010.

Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga (eds), *Handbook of Tableau Methods*, Springer, 1999.

Melvin Fitting & Richard L. Mendelsohn, *First-Order Modal Logic*, Dordrecht: Kluwer, 1998.

L.T.F. Gamut, *Logic, Language, and Meaning, Volume 2: intensional logic and logical grammar*, University of Chicago Press, 1991.

Hans Kamp and Uwe Reyle, *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, Vol. 42, Springer Science & Business Media, 2013.

Kate Kearns, *Semantics*, 2nd edition, Palgrave Macmillan, 2011.

Paul R. Kroeger, *Analyzing Meaning*, Language Science Press, 2019.

Stanley Peters and Dag Westerståhl, *Quantifiers in Language and Logic*, Oxford University Press, 2006.

Dag Prawitz, *Natural Deduction. A Proof-Theoretical Study*, Almqvist & Wiksell, 1965. Neuauflage 2006, Dover Publications.

Willard V. Quine, Quantifiers and propositional attitudes, *The Journal of Philosophy* 53(5) (1956), 177–187.

Raymond M. Smullyan, *First-Order Logic*, Ergebnisse der Mathematik und ihrer Grenzgebiete, Bd. 43, Springer, 1968. Reprinted by Dover Publications, 1995.

Alan M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, Series 2, **42** (1936), 230–265.

Dag Westerståhl, Generalized quantifiers, in: Edward N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy* (Winter 2019 Edition), https://plato.stanford.edu/archives/win2019/entries/generalized-quantifiers.

Thomas E. Zimmerman and Wolfang Sternefeld, *Introduction to Semantics: An Essential guide to the Composition of Meaning*, De Gruyter, 2013.

A Outputs generated with pyPL

A.1 Model checking for first-order logic

Structure $\mathcal{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ with

$\mathcal{D} = \{\text{John}, \text{Lea}, \text{Mary}, \text{Peter}, \text{Susan}\}$

$\mathcal{I}$:

lea	$\mapsto$ Lea
mary	$\mapsto$ Mary
susan	$\mapsto$ Susan
Happy	$\mapsto \{\langle \text{John} \rangle, \langle \text{Lea} \rangle, \langle \text{Peter} \rangle, \langle \text{Susan} \rangle\}$
Man	$\mapsto \{\langle \text{John} \rangle, \langle \text{Peter} \rangle\}$
Woman	$\mapsto \{\langle \text{Lea} \rangle, \langle \text{Mary} \rangle, \langle \text{Susan} \rangle\}$
Love	$\mapsto \{\langle \text{John}, \text{Peter} \rangle, \langle \text{Lea}, \text{Peter} \rangle, \langle \text{Mary}, \text{John} \rangle, \langle \text{Peter}, \text{Lea} \rangle, \langle \text{Susan}, \text{John} \rangle, \langle \text{Susan}, \text{Peter} \rangle\}$

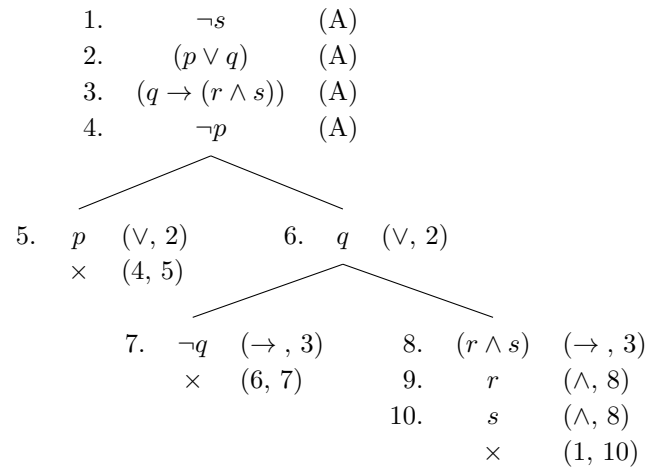
$[\![\forall x(\mathit{Woman}(x) \rightarrow \exists y(\mathit{Man}(y) \wedge \mathit{Love}(x, y)))]\!]^{\mathcal{S}}$
= True

$[\![\forall x(\mathit{Man}(x) \rightarrow \exists y(\mathit{Woman}(y) \wedge \mathit{Love}(x, y)))]\!]^{\mathcal{S}}$
= False

A.2 Tableau for a valid inference

You are using proof search for classical propositional logic with local validity.

Tableau for
$(p \vee q)$,
$(q \to (r \wedge s))$,
$\neg p$
$\vDash s$:



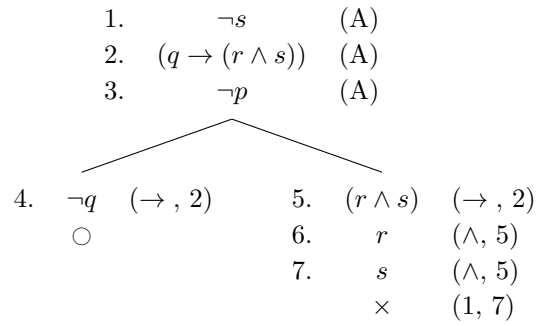
The tableau is closed:
The inference is valid.

This computation took 0.0018 seconds, 3 branches and 10 nodes.

A.3 Tableau for an invalid inference

You are using proof search for classical propositional logic with local validity.

Tableau for
$(q \to (r \wedge s))$,
$\neg p$
$\models s$:



The tableau is open:
The inference is invalid.

Countermodels:

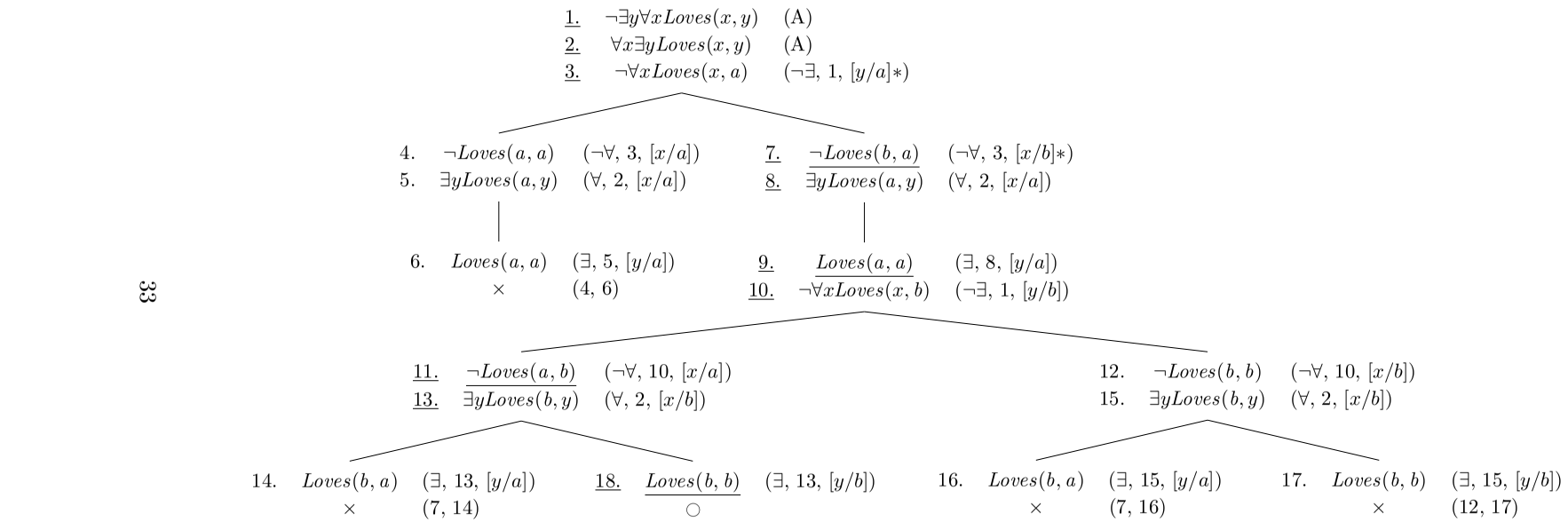
Structure $\mathcal{S}_4 = \langle \mathcal{V}_4 \rangle$ with
$\mathcal{V}_4 : p \mapsto 0, q \mapsto 0, s \mapsto 0$

This computation took 0.0013 seconds, 2 branches and 7 nodes.

A.4 Model generation for structural ambiguity

You are using countermodel generation for classical predicate logic and local validity.

Tableau for
$\forall x \exists y Loves(x, y)$
$\not\models \exists y \forall x Loves(x, y)$:



The tableau is open:
The inference is invalid.

Countermodels:

Structure $\mathcal{S}_{18} = \langle \mathcal{D}_{18}, \mathcal{I}_{18} \rangle$ with
$\mathcal{D}_{18} = \{\mathrm{a}, \mathrm{b}\}$
$\mathcal{I}_{18}$: $Loves \mapsto \{\langle \mathrm{a}, \mathrm{a} \rangle, \langle \mathrm{b}, \mathrm{b} \rangle\}$

This computation took 0.0124 seconds, 5 branches and 18 nodes.

A.6 Theorem proving for modal logic with constant domains

You are using proof search for classical modal predicate logic with constant domains and local validity in a K frame.

Tableau for
$\Diamond\exists x C(x)$
$\vDash \exists x \Diamond C(x)$:

1.	w_1	$\neg\exists x\Diamond C(x)$	(A)
2.	w_1	$\Diamond\exists x C(x)$	(A)
3.	w_1	$\neg\Diamond C(a)$	($\neg\exists$, 1, $[x/a]*$)
4.	w_2	$\exists x C(x)$	($\Diamond$, 2, $\langle w_1,w_2\rangle*$)
5.	w_1	$\neg C(a)$	($\neg\Diamond$, 3, $\langle w_1,w_1\rangle*$)
6.	w_2	$C(b)$	($\exists$, 4, $[x/b]*$)
7.	w_1	$\neg\Diamond C(b)$	($\neg\exists$, 1, $[x/b]$)
8.	w_1	$\neg C(b)$	($\neg\Diamond$, 7, $\langle w_1,w_1\rangle$)
9.	w_3	$\exists x C(x)$	($\Diamond$, 2, $\langle w_1,w_3\rangle*$)
10.	w_3	$C(c)$	($\exists$, 9, $[x/c]*$)
11.	w_2	$\neg C(a)$	($\neg\Diamond$, 3, $\langle w_1,w_2\rangle$)
12.	w_2	$C(d)$	($\exists$, 4, $[x/d]*$)
13.	w_2	$\neg C(b)$	($\neg\Diamond$, 7, $\langle w_1,w_2\rangle$)
		$\times$	(6, 13)

The tableau is closed:
The inference is valid.

This computation took 0.009 seconds, 1 branch and 13 nodes.

A.7 Theorem proving for modal logic with varying domains

You are using countermodel generation for classical modal predicate logic with varying domains and local validity in a K frame.

Tableau for
$\Diamond \exists x C(x)$
$\not\models \exists x \Diamond C(x)$:

The tableau is open:
The inference is invalid.

Countermodels:

Structure $\mathcal{S}_{14} = \langle \mathcal{W}_{14}, \mathcal{R}_{14}, \mathcal{D}_{14}, \mathcal{I}_{14} \rangle$ with
$\mathcal{W}_{14} = \{w_1, w_2\}$
$\mathcal{R}_{14} = \{\langle w_1, w_1\rangle, \langle w_1, w_2\rangle\}$
$\mathcal{D}_{14}$: $w_1 \mapsto \{\text{a}\}$
$w_2 \mapsto \{\text{b}\}$
$\mathcal{I}_{14}$: $C \mapsto w_1 \mapsto \{\}$
$w_2 \mapsto \{\langle \text{b} \rangle\}$

This computation took 0.0044 seconds, 4 branches and 14 nodes.

A.8 Model checking for intensional contexts

Structure $\mathcal{S} = \langle \mathcal{W}, \mathcal{R}, \mathcal{D}, \mathcal{I} \rangle$ with

$\mathcal{W} = \{w_0, w_1\}$

$\mathcal{R} = \{\}$

$\mathcal{D} = \{\text{DonaldTrump}, \text{JoeBiden}, \text{Mary}\}$

$\mathcal{I}$:

biden	$\mapsto$	w_0	$\mapsto$	JoeBiden
		w_1	$\mapsto$	JoeBiden
mary	$\mapsto$	w_0	$\mapsto$	Mary
		w_1	$\mapsto$	Mary
president	$\mapsto$	w_0	$\mapsto$	JoeBiden
		w_1	$\mapsto$	DonaldTrump
trump	$\mapsto$	w_0	$\mapsto$	DonaldTrump
		w_1	$\mapsto$	DonaldTrump
Democrat	$\mapsto$	w_0	$\mapsto$	$\{\langle \text{JoeBiden} \rangle\}$
		w_1	$\mapsto$	$\{\langle \text{JoeBiden} \rangle\}$
Republican	$\mapsto$	w_0	$\mapsto$	$\{\langle \text{DonaldTrump} \rangle\}$
		w_1	$\mapsto$	$\{\langle \text{DonaldTrump} \rangle\}$
Believe	$\mapsto$	w_0	$\mapsto$	$\{\langle \text{Mary}, ((w_1, True), (w_0, True)) \rangle\}$
		w_1	$\mapsto$	$\{\langle \text{Mary}, ((w_1, True), (w_0, True)) \rangle\}$

$[\![(biden = president)]\!]^{\mathcal{S}, w_0}$
= True

$[\![Believe(mary, {}^{\wedge} Democrat(biden))]\!]^{\mathcal{S}, w_0}$
= True

$[\![Believe(mary, {}^{\wedge} Democrat(president))]\!]^{\mathcal{S}, w_0}$
= False

A.9 Model checking for generalized quantifiers

Structure $\mathcal{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ with

$\mathcal{D} = \{\text{John}, \text{Lea}, \text{Mary}, \text{Peter}, \text{Susan}\}$

$\mathcal{I}$:

lea	$\mapsto$ Lea
mary	$\mapsto$ Mary
susan	$\mapsto$ Susan
Happy	$\mapsto \{\langle\text{John}\rangle, \langle\text{Lea}\rangle, \langle\text{Peter}\rangle, \langle\text{Susan}\rangle\}$
Man	$\mapsto \{\langle\text{John}\rangle, \langle\text{Peter}\rangle\}$
Woman	$\mapsto \{\langle\text{Lea}\rangle, \langle\text{Mary}\rangle, \langle\text{Susan}\rangle\}$
Love	$\mapsto \{\langle\text{John}, \text{Peter}\rangle, \langle\text{Lea}, \text{Peter}\rangle, \langle\text{Mary}, \text{John}\rangle, \langle\text{Peter}, \text{Lea}\rangle, \langle\text{Susan}, \text{John}\rangle, \langle\text{Susan}, \text{Peter}\rangle\}$

$[\![\text{most } x(\mathit{Woman}(x), \mathit{Happy}(x))]\!]^{\mathcal{S}}$
= True

$[\![\text{more } x(\mathit{Woman}(x), \mathit{Man}(x), \mathit{Happy}(x))]\!]^{\mathcal{S}}$
= False