Recap: Tableaux

Extension to FOL
  Without Unification
  Unification
  With Unification

Undecidability

Model Building
  The Big Picture
  Main Approaches
  Tableau-Based Methods
  OletinMB and PRIDAS

Conclusion

First Order Tableaux

Johannes Dellert

Universität Tübingen, Seminar für Sprachwissenschaft

11.05.2010

Outline

Recap: Tableaux

Extension to FOL

Undecidability

Model Building

Conclusion

Tableau Rules for Propositional Logic

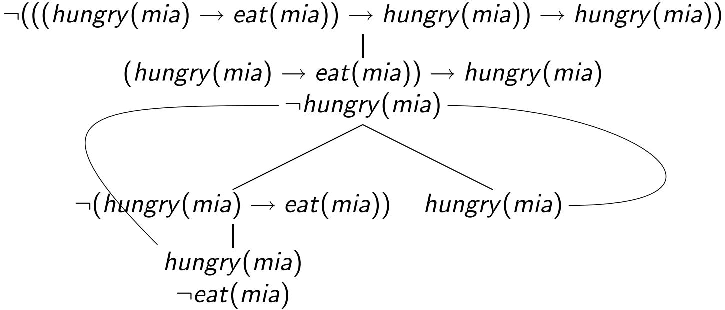
- the notation I use encodes polarities directly via negation
- as a result, we need only seven rules, and the tableaux only contain formulae of propositional logic
- branches are closed not via pairs $(T\phi, F\phi)$, but $(\phi, \neg\phi)$

$$\frac{\neg\neg\varphi}{\varphi} \qquad \frac{\varphi \land \psi}{\begin{array}{c}\varphi \\ \psi\end{array}} \qquad \frac{\neg(\varphi \lor \psi)}{\begin{array}{c}\neg\varphi \\ \neg\psi\end{array}} \qquad \frac{\neg(\varphi \rightarrow \psi)}{\begin{array}{c}\varphi \\ \neg\psi\end{array}}$$

$$\frac{\varphi \lor \psi}{\varphi \qquad\qquad \psi} \qquad \frac{\neg(\varphi \land \psi)}{\neg\varphi \qquad\qquad \neg\psi} \qquad \frac{\varphi \rightarrow \psi}{\neg\varphi \qquad\qquad \psi}$$

An example

Sentence 1: If Mia eats when she is hungry, Mia is hungry now.
Sentence 2: Mia is hungry.
Question: Is Sentence 2 informative?



$$\neg(((hungry(mia) \rightarrow eat(mia)) \rightarrow hungry(mia)) \rightarrow hungry(mia))$$

$$(hungry(mia) \rightarrow eat(mia)) \rightarrow hungry(mia)$$

$$\neg hungry(mia)$$

$$\neg(hungry(mia) \rightarrow eat(mia)) \qquad hungry(mia)$$

$$hungry(mia)$$

$$\neg eat(mia)$$

Answer: Sentence 1 follows from Sentence 2 ⇒ not informative.

Outline

Recap: Tableaux

Extension to FOL
- Without Unification
- Unification
- With Unification

Undecidability

Model Building

Conclusion

The Problem

Propositional Tableaux

- deterministic rules break down formulae into ever smaller pieces; this ensures termination together with the fact that
- every tableau node only needs to be processed once
- contradictions are always detected at the literal level

Treatment of Quantifiers

- refutation proofs will need to talk about elements of a universe that we do not (yet) have names for
- there is no obvious way of resolving an existential quantifier using finite branching if the universe is potentially infinite
- a universal quantifier can only be completely replaced if we talk about some finite universe that will not be expanded

The Intuitive Idea

- when resolving universal quantification, our task is to non-deterministically guess an instantiation that helps us to find a contradiction
- "If everyone must die, this also holds for me, so I can conclude I am going to die." $\forall x.die(x) \Rightarrow die(i)$
- since we want to refute the top formula, we try to choose instantiations that lead to closed branches
- when resolving existential quantification, we introduce new constant symbols in a process called **skolemization**
- "A robber must exist. None of the entities I have names for is a robber, so I introduce a new robber and call it e.g. _324."

Tableau Rules without Unification

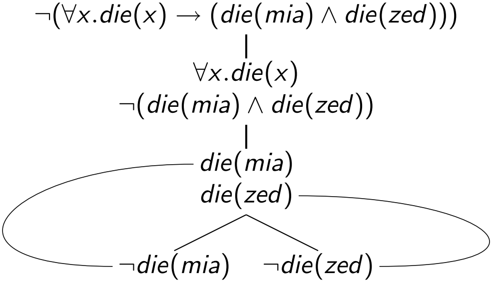
$$(\forall)\frac{\forall x.\gamma(x)}{\gamma(t)},\ \textit{where } t \textit{ is an arbitrary ground term}$$

$$(\neg\exists)\frac{\neg\exists x.\gamma(x)}{\neg\gamma(t)},\ \textit{where } t \textit{ is an arbitrary ground term}$$

$$(\exists)\frac{\exists x.\delta(x)}{\delta(c)},\ \textit{where } c \textit{ is a new constant symbol}$$

$$(\neg\forall)\frac{\neg\forall x.\delta(x)}{\neg\delta(c)},\ \textit{where } c \textit{ is a new constant symbol}$$

An Example



$\neg(\forall x.die(x) \rightarrow (die(mia) \wedge die(zed)))$

$\forall x.die(x)$

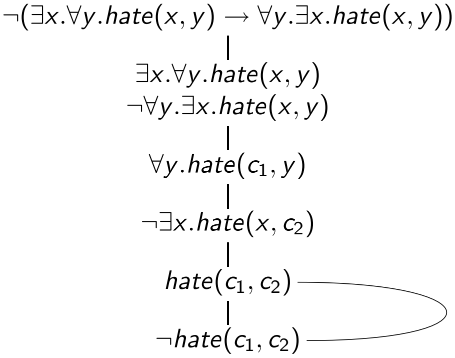
$\neg(die(mia) \wedge die(zed))$

$die(mia)$

$die(zed)$

$\neg die(mia)$ $\neg die(zed)$

Another Example



$$\neg(\exists x.\forall y.hate(x,y) \rightarrow \forall y.\exists x.hate(x,y))$$

$$\exists x.\forall y.hate(x,y)$$

$$\neg\forall y.\exists x.hate(x,y)$$

$$\forall y.hate(c_1,y)$$

$$\neg\exists x.hate(x,c_2)$$

$$hate(c_1,c_2)$$

$$\neg hate(c_1,c_2)$$

The Problem of Non-Determinism

- the main problem of our approach is that the choice of the ground terms to be substituted under the universal quantification rule is non-deterministic
- as humans, we rely on our intuitions and often manage to correctly guess the most promising substitutions
- a computer obviously does not have this kind of intuition, leading to a possibly exponential amount of wasted time while exploring branches that will clearly never be closed
- a promising approach to this problem is to delay the substitution decisions until we know more about the interdependence of our instantiation choices

Substitutions

Definition

A **substitution** is a function σ that maps variables to terms of a FO language. We will write $x\sigma$ instead of $\sigma(x)$ to denote the value of a variable x under σ.

Special Notation for Finite Substitutions

$\sigma = \{x_1/\tau_1, \dots, x_n/\tau_n\}$ means that for $i = 1, \dots, n$, distinct variables x_i are mapped to terms τ_i with $\tau_i \neq x_i$.

Substitutions on Terms

Let σ be a substitution and τ a term. Then

- $\tau = x \in Var \Rightarrow \tau\sigma = x\sigma$ or undefined
- $\tau = c \in Const \Rightarrow \tau\sigma = \tau$
- $\tau = f(\tau_1, \dots, \tau_n) \Rightarrow [f(\tau_1, \dots, \tau_n)]\sigma = f(\tau_1\sigma, \dots, \tau_n\sigma)$

Special Notation for x-Variants of Substitutions

σ_x is exactly like σ except that $x\sigma_x = x$

Substitutions

Substitutions on Formulae

- $R(\tau_1, \ldots, \tau_n)$ atomic $\Rightarrow [R(\tau_1, \ldots, \tau_n)]\sigma = R(\tau_1\sigma, \ldots, \tau_n\sigma)$
- $[\neg\varphi]\sigma = \neg[\varphi\sigma]$
- $[\varphi \circ \psi]\sigma = [\varphi\sigma] \circ [\psi\sigma]$ for $\circ \in \{\vee, \wedge, \rightarrow\}$
- $[\forall x.\varphi]\sigma = \forall x.[\varphi\sigma_x]$ and $[\exists x.\varphi]\sigma = \exists x.[\varphi\sigma_x]$

Substitution on Tableaux

If σ is a substitution and $\mathcal{T}$ a tableau, then $\mathcal{T}\sigma$ is the tableau obtained by replacing every formula φ in $\mathcal{T}$ by $\varphi\sigma$.

Composition of Substitutions

If σ_1 and σ_2 are substitutions, then $\sigma_1\sigma_2$ with $x\sigma_1\sigma_2 := (x\sigma_1)\sigma_2$ is again a substitution.

Generality Order on Substitutions

- assume we want to **unify** $f(c, y, w)$ and $f(x, y, g(z))$
- this could be achieved using $\sigma_1 := \{x/c, w/g(z)\}$:
  - $[f(c, y, w)]\sigma_1 = f(c, y, g(z))$
- but we could also use $\sigma_2 := \{x/c, w/g(z), y/h(u, x)\}$ ::
  - $[f(c, y, w)]\sigma_2 = f(c, h(u, x), g(z))$
  - $[f(x, y, g(z))]\sigma_2 = f(c, h(u, x), g(z))$
- σ_1 is our preferred choice because we are not over-committing ourselves, only substituting as much as strictly necessary

Generality of Substitutions

σ_1 is more general than σ_2 $:\Longleftrightarrow$
there is a substitution θ such that $\sigma_2 = \sigma_1\theta$

Unification

Unification

Let τ_1 and τ_2 be terms.

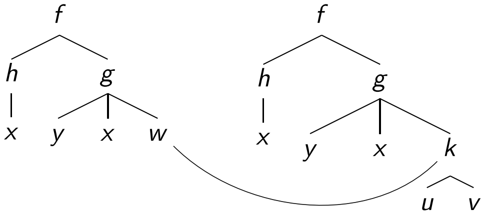
1) A substitution σ is a **unifier** of τ_1 and τ_2 $:\Leftrightarrow \tau_1\sigma = \tau_2\sigma$.

2) τ_1 and τ_2 are **unifiable** $:\Leftrightarrow$ τ_1 and τ_2 have a unifier σ

3) σ is a **most general unifier (MGU)** of τ_1 and τ_2 $:\Leftrightarrow$ σ is a unifier for τ_1 and τ_2, and it is more general than any other unifier for τ_1 and τ_2.

- the unifier of two terms can be seen as a set of constraints that must be fulfilled if the terms are to stay the same
- ideally, unifiers are **idempotent** ($\sigma\sigma = \sigma$), because this allows to incrementally solve the simultaneous unification problem
- we can use such a unifier to share re-usable information among the branches of a tableau
- this avoids a lot of work because the tableau algorithm can use information from other failed branches

A Naive Unification Algorithm



```
public Substitution unify(Term t1, Term t2) {
  Substitution s = new Substitution();
  while (!equals(t1.apply(s),t2.apply(s))) {
    Pair<Term> d1d2 = getDisagrPair(t1.app(s),t2.app(s))
    if (!isSimpleDisagreementPair(d1d2)) {
      throw new NotUnifiableException(d1d2);
    } else {
      s.addRelevantRepairFor(d1d2);
    }
  }
  return s;
}
```

Importance of Unification

- historically, only unification made first-order theorem proving feasible; still the most important concept in theorem proving
- fast implementations are the most crucial component of logic programming and many constraint programming systems
- Prolog heavily relies on efficient unification: goal calls usually require unification for each argument
- unification theory has become a special branch of science
- state-of-the-art unification technology is highly optimized and very fast, but extremely unintuitive and hard to understand

Skolemization

- in our unification-based tableau method, we will resolve existential quantification by means of **skolemization**
- this means we introduce **Skolem functions** for each entity that depend on the free variables of the existential formula
- in the first version, we only had **Skolem constants** - but also no free variables, so we have a genuine generalization
- why is skolemization crucial?
  - $\exists y(\neg R(x,y)) \land R(x,x) \Rightarrow \neg R(x,w) \land R(x,x)$
  - unification $\Rightarrow \neg R(x,x)$ and $R(x,x)$ on one branch!
  - but the formula is satisfiable!
  - with skolemization: $\neg R(x,s(x)) \land R(x,x)$, unification prevented because x occurs in $s(x)$

Free-Variable Tableau Rules

$$(\forall)\frac{\forall x.\gamma(x)}{\gamma(x')},\ x' \textit{ a variable not occurring elsewhere in the tableau}$$

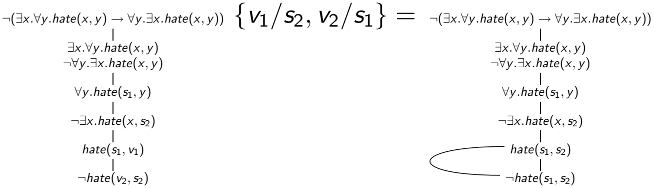
$$(\neg\exists)\frac{\neg\exists x.\gamma(x)}{\neg\gamma(x')},\ x' \textit{ a variable not occurring elsewhere in the tableau}$$

$$(\exists)\frac{\exists x.\delta(x)}{\delta(f(x_1,...,x_n))},\ f \textit{ new function symbol},\ x_i \textit{ free variables in } \delta$$

$$(\neg\forall)\frac{\neg\forall x.\delta(x)}{\neg\delta(f(x_1,...,x_n))},\ f \textit{ new function symbol},\ x_i \textit{ free variables in } \delta$$

Back to the Example

- for each pair of literals on a branch, we compute the MGU
- if the MGU exists, we apply it to the entire tableau, hopefully closing many branches, as in the following example
- $MGU(hate(s_1, v_1), hate(v_2, s_2)) = \{v_1/s_2, v_2/s_1\}$, we apply it to the tableau and close a branch:



$\neg(\exists x.\forall y.hate(x,y) \rightarrow \forall y.\exists x.hate(x,y))$

$\exists x.\forall y.hate(x,y)$

$\neg\forall y.\exists x.hate(x,y)$

$\forall y.hate(s_1,y)$

$\neg\exists x.hate(x,s_2)$

$hate(s_1,v_1)$

$\neg hate(v_2,s_2)$

$$\{v_1/s_2, v_2/s_1\} =$$

$\neg(\exists x.\forall y.hate(x,y) \rightarrow \forall y.\exists x.hate(x,y))$

$\exists x.\forall y.hate(x,y)$

$\neg\forall y.\exists x.hate(x,y)$

$\forall y.hate(s_1,y)$

$\neg\exists x.hate(x,s_2)$

$hate(s_1,s_2)$

$\neg hate(s_1,s_2)$

Implementation Issues

- with unification, the whole tableau sometimes needs to be changed, so we have to keep it all in memory
- the branches are not independent of each other any more, risks of overcommitting remain

\+ knowledge of how to produce a contradiction can often be exploited at many other places

\+ we can delay instantiation decisions and do not have to explore so many failing branches

Outline

Recap: Tableaux

Extension to FOL

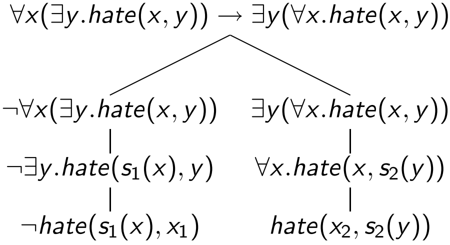
Undecidability

Model Building

Conclusion

Trying to prove invalidity

Show that $\forall x(\exists y.hate(x,y)) \rightarrow \exists y(\forall x.hate(y,x))$ is invalid:



$$\forall x(\exists y.hate(x,y)) \rightarrow \exists y(\forall x.hate(x,y))$$

Left branch:

$$\neg\forall x(\exists y.hate(x,y))$$
$$\neg\exists y.hate(s_1(x),y)$$
$$\neg hate(s_1(x),x_1)$$

Right branch:

$$\exists y(\forall x.hate(x,y))$$
$$\forall x.hate(x,s_2(y))$$
$$hate(x_2,s_2(y))$$

What this Means

- we cannot reliably prove invalidity of a formula any more
- this means the tableau method can only give us a negative check for informativity, we do not really have a decider
- no other theorem prover for FOL can achieve that either
- it is possible to prove formally that FOL is undecidable:
  - devise a method to describe the halting problem for turing machines by formulae of FOL
  - we are thereby reducing FOL validity to the halting problem, which proves it is at most semi-decidable
- validity is a recursively enumerable (i.e. semi-decidable) problem (Chomsky Type 0)
- invalidity is not even recursively enumerable (!)

Outline

Recap: Tableaux

Extension to FOL

Undecidability

Model Building
- The Big Picture
- Main Approaches
- Tableau-Based Methods
- OletinMB and PRIDAS

Conclusion

Theorem Proving vs. Model Building

	consistency (= satisfiability)	informativity (= invalidity)
positive (**model building**)	$\exists\mathfrak{A} : \mathfrak{A} \models \varphi$ $\exists\mathfrak{A} : \mathfrak{A} \models \varphi$	$\exists\mathfrak{A} : \mathfrak{A} \models \neg\varphi$ $\neg\forall\mathfrak{A} : \mathfrak{A} \models \varphi$
negative (**theorem proving**)	$\neg\exists\mathfrak{A} : \mathfrak{A} \models \varphi$ $\forall\mathfrak{A} : \mathfrak{A} \models \neg\varphi$	$\forall\mathfrak{A} : \mathfrak{A} \models \varphi$ $\forall\mathfrak{A} : \mathfrak{A} \models \varphi$

- **theorem provers** can only decide whether a formulae holds in all models or in no model at all
- they are able to prove validity or unsatisfiability
- **model builders** possess the ability to exhibit specific models or counter models for some formulae
- this sometimes allows them to prove satisfiability or invalidity
- theorem prover proofs are proofs by contradiction
- model builder proofs are constructive in nature

Enumerative vs. Constructive

Enumerative Methods

- first introduced by MACE and SEM
- essentially generate candidate models and perform checking
- usually first-order logic is flattened to PL over finite domains
- much more than "trial and error": symmetries must be exploited and isomorphic models avoided
- implementations are highly optimized and often rely on complex constraint propagation schemes
- still only feasible for rather small models ($|\mathfrak{A} < 20|$)

Constructive Methods

- the method used by many experimental systems
- usually rely on saturated open branches in tableaux
- manage to build large models if the search space is not too heavily constrained; not good at "model finding"
- a lot more transparent even with optimizations, allow for customization and external guidance

Minimal vs. Non-Minimal

Minimal Model Building

- a model for a formula φ is minimal iff $\neg\exists$ smaller model for φ
- enumerative methods are usually incremental and can therefore only produce minimal models
- desired by the mathematician, the minimal structures licensed by axioms are often especially important
- not appropriate for our purposes because we generally do not assume entities to be identical by default

Non-Minimal Model Building

- not very well defined, for most formulae one can find models of arbitrary size and complexity
- non-minimal model building therefore requires a formalism to constrain the class of desired models
- not much work in the area, constraints are usually enforced by inflating theories with additional axioms
- big advantage: can avoid even considering models with certain properties at construction time

The Intuitive Idea

- What does it mean if we do not manage to close a particular branch during tableau construction?
  - we cannot refute the formula for certain choices of variable instantiations, we failed in trying to derive a contradiction
  - could it even be that the formula is true under the choices we made while arriving at that branch?
  - yes, if we have tried everything we could to close it
- Important Observations:
  - these so-called branch-saturating choices can be read off the literals along open branches of the tableau
  - the literals of a saturated open branch constitute an (underspecified) description of a model

A Simple Example

$$\exists x \exists y : rabbit(x) \wedge carrot(y) \wedge eat(x, y)$$

$$\exists y : rabbit(c_1) \wedge carrot(y) \wedge eat(c_1, y)$$

$$rabbit(c_1) \wedge carrot(c_2) \wedge eat(c_1, c_2)$$

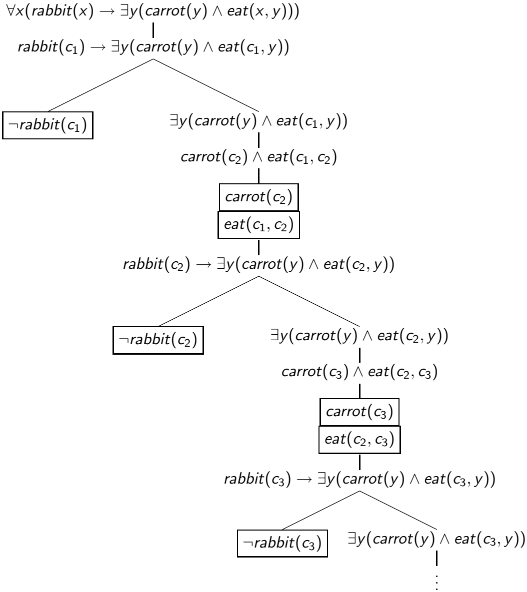
$$\boxed{rabbit(c_1)}$$

$$carrot(c_2) \wedge eat(c_1, c_2)$$

$$\boxed{carrot(c_2)}$$

$$\boxed{eat(c_1, c_2)}$$

Another Simple Example



$\forall x(rabbit(x) \rightarrow \exists y(carrot(y) \wedge eat(x,y)))$

$rabbit(c_1) \rightarrow \exists y(carrot(y) \wedge eat(c_1,y))$

$\boxed{\neg rabbit(c_1)}$

$\exists y(carrot(y) \wedge eat(c_1,y))$

$carrot(c_2) \wedge eat(c_1,c_2)$

$\boxed{carrot(c_2)}$

$\boxed{eat(c_1,c_2)}$

$rabbit(c_2) \rightarrow \exists y(carrot(y) \wedge eat(c_2,y))$

$\boxed{\neg rabbit(c_2)}$

$\exists y(carrot(y) \wedge eat(c_2,y))$

$carrot(c_3) \wedge eat(c_2,c_3)$

$\boxed{carrot(c_3)}$

$\boxed{eat(c_2,c_3)}$

$rabbit(c_3) \rightarrow \exists y(carrot(y) \wedge eat(c_3,y))$

$\boxed{\neg rabbit(c_3)}$

$\exists y(carrot(y) \wedge eat(c_3,y))$

$\vdots$

A Problematic Example

$$\forall x \exists y (love(x, y))$$

$$\exists y (love(c_1, y))$$

$$\boxed{love(c_1, c_2)}$$

$$\exists y (love(c_2, y))$$

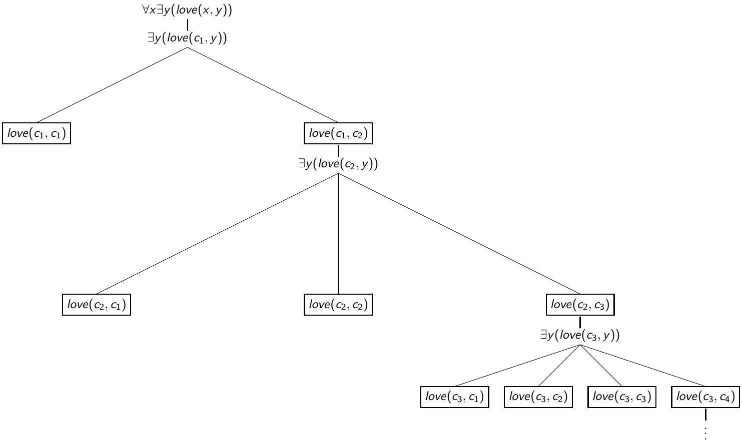
$$\boxed{love(c_2, c_3)}$$

$$\exists y (love(c_3, y))$$

$$\boxed{love(c_3, c_4)}$$

$$\vdots$$

Solution: Identity Assumptions



$\forall x \exists y (love(x, y))$

$\exists y (love(c_1, y))$

$\boxed{love(c_1, c_1)}$ $\boxed{love(c_1, c_2)}$

$\exists y (love(c_2, y))$

$\boxed{love(c_2, c_1)}$ $\boxed{love(c_2, c_2)}$ $\boxed{love(c_2, c_3)}$

$\exists y (love(c_3, y))$

$\boxed{love(c_3, c_1)}$ $\boxed{love(c_3, c_2)}$ $\boxed{love(c_3, c_3)}$ $\boxed{love(c_3, c_4)}$

$\vdots$

Challenge: Search Space Exploration

- Important Observations:
  - we do what we avoided in theorem proving: trying out identity assumptions between constants
  - this is because our goal is exactly the opposite: we want to find branches we can NOT close
  - in a sense, we are over-committing in order to inspect special cases (= models) of a class of structures
- The Challenge of Search Space Exploration
  - our problem is that unification does not help us any more to make good assumptions, as it guids us towards contradictions
  - we are thrown back to non-deterministically exploring possibly vast search spaces without having a clue about their structure
- Possible Solutions
  - optimization techniques can help us to avoid running into the same contradiction very often
  - domain-specific knowledge can help us to nudge search space traversal into a promising direction

OletinMB: A Novel Model Builder

- OletinMB is my model builder imlementation that I designed as a term project in order to explore the potential of non-minimal model building for linguistic purposes
- “oletin” roughly translates as “assuming device”
- written entirely in Java for efficiency and portability reasons, minimal model building performance can compete with MACE in the areas it is specialized for
- can mimick MACE’s behavior, allowing it to be tested against MACE in all applications without further glue code
- crucial feature is an open interface for allowing external guidance of search space traversal

PRIDAS: Prioritized Identity Assumptions

- the main hook within OletinMB for search space control:
- the priority ordering of variable instantiations in the existential rules can be defined as a function depending on internal and external factors
- possible internal factors: topological properties of the tableau, propagated constraints, occurrence patterns
- possible external factors: any weighting function (e.g. wordnet similarity between predicate names, a fuzzy word knowledge database encoded as a weighted decision network)
- the simplest possible PRIDAS functions:
    - always rank the introduction of a new constant lowest: minimal model building, mimicking MACE's behavior
    - always try introducing a new constant first: least commitment and highest generality, but with increased risks of non-termination

Preliminary Results

- on formulae representing short newspaper texts (as in the RTE challenge), the satisifiability task is generally accomplished very well
- not yet always fast enough for proving informativity: if search spaces get very sparse, traversal tends to get too inefficient
- world knowledge is no longer a strict necessity for constructing sensible models; for many sentences, the results without are as good as MACE's results with world knowledge

Current Work

- optimizations to make OletinMB useful for positive informativity checking
- experiments with more complex PRIDAS functions involving external sources of knowledge
- quantitative tests against the Nutcracker RTE system to see whether the improvements are statistically significant
- further research into implicit constraints on linguistic models

Outline

Recap: Tableaux

Extension to FOL

Undecidability

Model Building

Conclusion

Important Points

First-Order Theorem Proving

- is an extremely hard and generally undecidable problem that has spawned a highly developed branch of science
- unification is the crucial concept for limiting non-determinism during proof search, TP performance depends largely on efficiency of unification implementations
- tableau methods can also be applied without unification if the structure of the search space is well-understood

Model Building

- complements theorem proving by providing examples and counter-examples
- is an even harder task than theorem proving, completely unfeasible on a larger scale, enumeration is state of the art
- can be done with highly adaptable systems based on tableau methods if one specializes on certain classes of formulae

Conclusion

Thank you.