Automated Deduction Techniques for Knowledge Representation Applications

Peter Baumgartner

Max-Planck-Institute for Informatics

The Big Picture

Knowledge Base

Ontologies
- OWL DL
  (Tambis, Wine, Galen)
- First-Order
  (SUMO/MILO, OpenCyc)
- FrameNet

Rules (SWRL)

Data (ABox)

Reasoning

Tasks
- TBox: (Un)satisfiability,Subsumption
- ABox: Instance, Retrieve
- General entailment tasks

Theorem provers for:
- Classical FO (ME: Darwin)
- FO with Default Negation
  (Hyper Tableaux: KRHyper)

Robust Reasoning Services?

- **Issues:** undecidable logic, model computation, equality, size
- **Approach:** transformation of KB tailored to exploit prover features

Contents

- Transforming the knowledge base for reasoning
  - Transformation of OWL to clause logic: about equality
  - Treating equality
  - Blocking
- Theorem proving
  - KRHyper model generation prover
  - Experimental evaluation
- Rules: an application for reasoning on FrameNet

Transformation of OWL to Clause Logic

- We use the WonderWeb OWL API to get FO Syntax first
- Then apply standard clause normalform trafo (except for "blocking")
- Equality comes in, e.g., for
  - **nominals** ("oneOf")

$$\text{WhiteLoire} \sqsubseteq \forall\, \text{madeFromGrape} \,.\, \text{Sauvignon} \sqcup \text{Chenin} \sqcup \text{Pinot}$$

$$\text{WhiteLoire}(x) \wedge \text{madeFromGrape}(x, y) \Rightarrow y = \text{Sauvignon} \vee y = \text{Chenin} \vee y = \text{Pinot}$$

  - **cardinality restrictions**

$$\text{Cation} \sqsubseteq\ \leq 4\ \text{hasCharge}$$

$$\text{Cation}(x) \wedge \text{hasCharge}(x, x1) \wedge \cdots \wedge \text{hasCharge}(x, x5) \Rightarrow x1 = x2 \vee x1 = x3 \vee \cdots \vee x4 = x5$$

- -> Need an (efficient) way to treat **equality**

Equality

- **Option1:** use equality axioms
  But substitution axioms $x = y \Rightarrow f(x) = f(y)$ - cumbersome
- **Option 2:** use a (resolution) prover with built-in equality
  But how to extract a model from a failed resolution proof?
  We focus on systems for model generation
- **Option 3**: Transform equality away a la Brand's transformation
  Problem: Brand's Transformation is not "efficient enough"
  Solution: Use a suitable, modified Brand transformation

Brand's Transformation Revisited

Extension of Brand's Method: UNA for constants (optional)

Add $\neg(a = b)$ for all different constants a and b

Modified Flattening

Given: $P(f(x)) \leftarrow f(g(a)) = h(a, x)$

Brand: $P(z1) \leftarrow f(z2) = z3,\ h(z4, x) = z3,\ f(x) = z1,\ g(z4) = z2,\ a = z4$

Our trafo: $P(f(x)) \leftarrow f(z1) = h(a, x),\ g(a) = z1$

A clause is flatt iff all proper subterms are constants or variables

Our Transformation

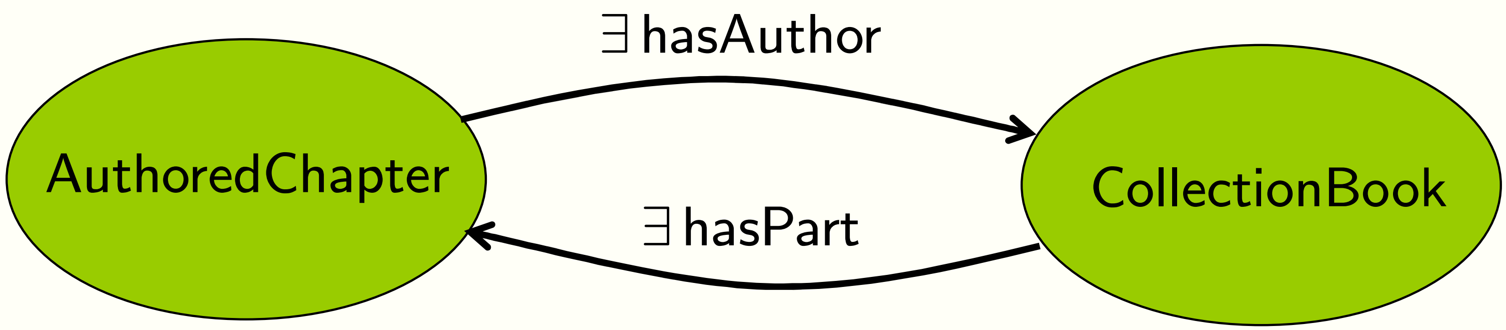
- modified flattening
- add equivalence relation axioms for =
- add predicate substitution axioms $P(y) \leftarrow P(x), x = y$

It works much better in practice!

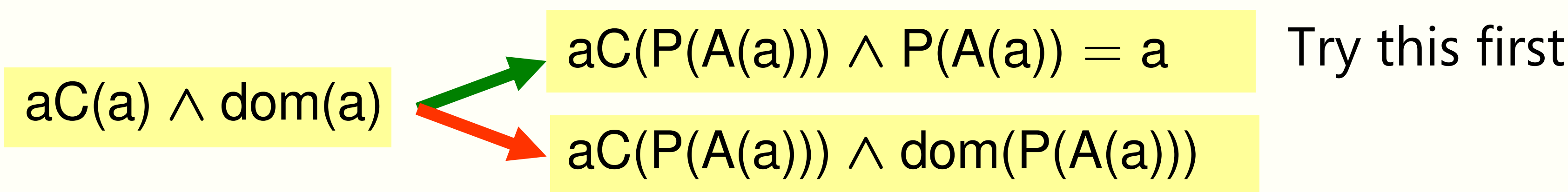
Blocking

- **Problem:** Termination in case of satisfiable input
  Specifically: cyclic definitions in TBox
  Example from Tambis KB:

TBox



- **Solution:** Learn from blocking technique from description logics
  "Re-use" previously introduced individual to satisfy exist-quantifier
  Here: encode search for model with finite domain in clause logic:



- **Issue:** Make it work fast: don't be too ambitious on speculating

KRHyper

- **Semantics**
  - Classical predicate logic (refutational complete)
  - Stable models of normal programs (with transformation)
  - Possible models for disjunctive programs (with transformation)

- **Efficient Implementation** (in Ocaml):

  Transitive closure of 16.000 facts -> 217.000 facts:

KRHyper:	17 sec,	63 Mb
Otter (pos. hyperres)	37 min,	124 Mb
Compiling SATCHMO:	2:14 h,	271 Mb
smodels:	-	-

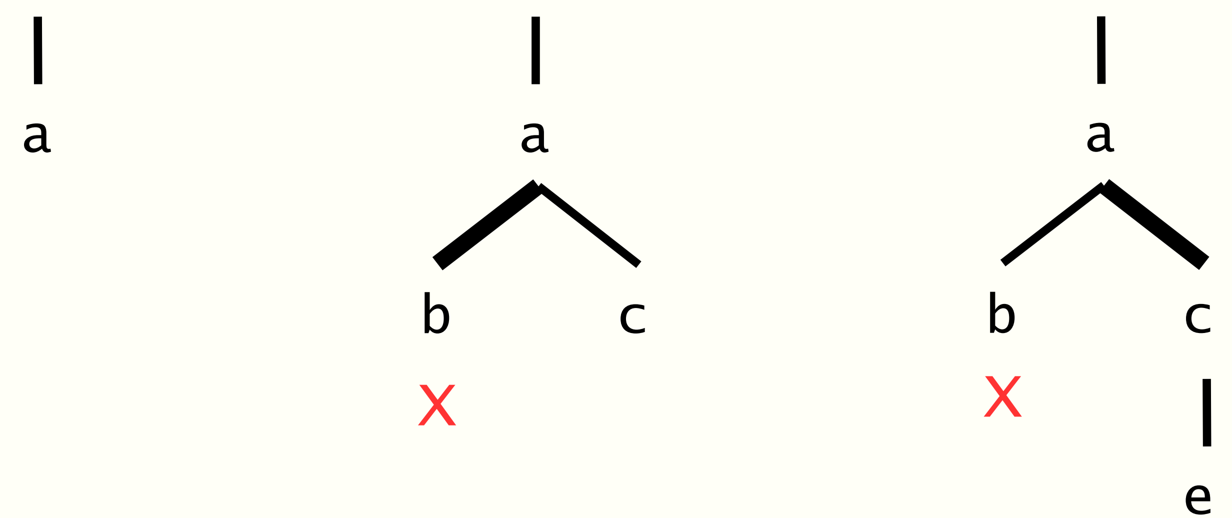
- **User manual**
- **Proof tree output**

Computing Models with KRHyper

- Disjunctive logic programs
- Stratified default negation

```
a.                (1)
b ; c :- a.       (2)
a ; d :- c.       (3)
false :- a,b.     (4)
```

```
e :- c, not d.  (5)
```



{} ⊭ (1) {a} ⊭ (2) {a,b} ⊭ (4) X {a,c} ⊨ (1)-(4)

- Variant for predicate logic
- Extensions: minimal models, abduction, **default negation**

Experimental Evaluation

OWL Test Cases

System	Consistent (56)	Inconsistent (72)	Entailment (111)
KRHyper with blocking	86%	89%	93%
KRHyper w/o blocking	79%	94%	93%
Fact	42%	85%	7%
Hoolet	78%	94%	72%
FOWL	53%	4%	32%
Pellet	96%	98%	86%
Euler	0%	98%	100%
OWLP	50%	26%	53%
Cerebra	90%	59%	61%
Surnia	-	0%	13%
ConsVISor	77%	65%	-

Realistically Sized Ontologies

- **Tambis**
  - About chemical structures, functions, processes, etc within a cell
  - 345 concepts,107 roles
  - KRHyper: 2 sec per subsumption test
- **Wine**
  - Wine and food ontology, from the OWL test suite
  - 346 concepts, 16 roles, 150 GCIs, ABox
  - KRHyper: 80 sec / 3 sec per negative / positive subsumption test
- **Galen Common Reference Model**
  - Medical teminology ontology
  - big: 24.000 concepts, 913.000 relations, 400 GCIs, transitivity
  - KRHyper: 5 sec per subsumption test
- **OpenCyc**
  - 480.000 (simple) rules. Darwin: 60 sec for satisfiability

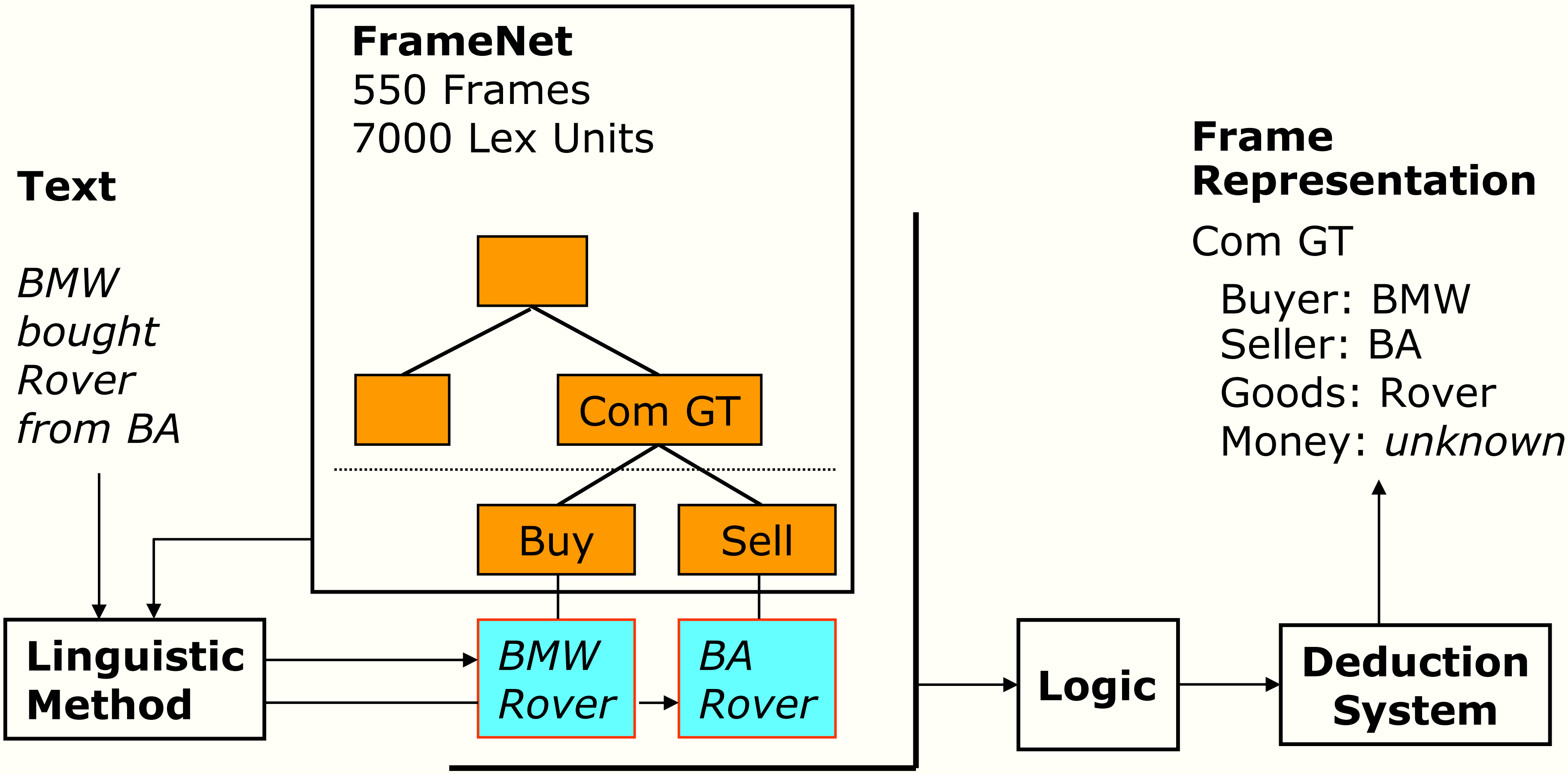
Rules

- Adding logic programming style **rules** is currently discussed in the Semantic Web context (SWRL and many others)
- **Example:**

  $$\text{HomeWorker}(x) \leftarrow \text{work}(x, y) \wedge \text{live}(x, z) \wedge \text{loc}(y, w) \wedge \text{loc}(z, w)$$

  Cannot be expressed in description logics
- Adding rules to the input language is trivial in approaches that transform ontologies to clause logic
- **Problem:** can simulate Role-Value maps, leading to undecidability
- **Rationale** of doing it nethertheless:
  - Better have only a semi-decision procedure than nothing
  - In many cases have termination nethertheless (with blocking)
  - Really useful in some applications

From Natural Language Text To Frame Representation



Work in Colaboration with Computer Linguistics Department (Prof. Pinkal)

Transfer of Role Fillers

(Slide by Gerd Fliedner)

FrameNet						Request Auftrag (noun)						
									Message			
	Receiving erhalten (verb)											
	Recipient				Donor	Theme						
PReDS	Preds erhalten (verb)											
	DSub Flugzeug#hersteller (noun)			PPMod von (praep)		DObj Auftrag (noun)						
					Arg Groß#britannien (noun)			PPModdef für (praep)				
									Arg Transport#flug#zeuge (noun)			
Topologie & Chunker	KS											
	VF		LK	MF							RK (simple)	
	NP			PP		NP		PP			Stem	
	DEF-ART	S		PRAEP	EIGEN	DEF-ART	S	PRAEP	KARD	S		
Text	Der	Flugzeughersteller	hat	von	Großbritannien	den	Auftrag	für	25	Transportflugzeuge	erhalten	.
Morph	der (PRON, ART) die (PRON, ART)	Flugzeug#hersteller (S) Flug#zeug#hersteller (S)	haben (V)	von (PRAEP, EIGEN)	Groß#britannien (S)	der (PRON, ART) die (ART)	Auftrag (S)	für (ADV, PRAEP)	25 (NUM)	Transport#flug#zeuge (S) Transport#flugzeug (S)	erhalten (A(PART), V)	. (PUNCT)

The plane manufacturer has from Great Britain the order for 25 transport planes received.

Task: Fill in the missing elements of „Request“ frame

Transfer of Role Fillers

The plane manufacturer has from Great Britain the order for 25 transport planes received.

Parsing gives **partially** filled FrameNet frame instances of „receive“ and „request“:

receive1:
```
receive
          target: „received“
           donor: „Great Britain“
       recipient: manufacturer1
           theme: request1
```

request1:
```
request
          target: „order“
         speaker: „Great Britain“
       addressee: manufacturer
         message: „transport plane“
```

- Transfer of role fillers done so far manually
- Can be done automatically. **By „model generation“**

Transfer of Role Fillers by Rules

```
receive1:  receive
                    target: „received“
                     donor: „Great Britain“
                 recipient: manufacturer1
                     theme: request1

request1:  request
                    target: „order“
                   speaker: „Great Britain“
                 addressee:
                   message: „transport plane“
```

Rules

```
speaker(Request, Donor) :-
        receive(Receive),
        donor(Receive, Donor),
        theme(Receive, Request),
        request(Request).
```

Facts

```
receive(receive1).
donor(receive1,
        „Great Britain“).
theme(receive1,request1).
request(request1).
```

Exploiting Nonmonotonic Negation: Default Values

Insert default value as a role filler in absence of specific information

```
receive1:  receive
                   target: „received“
                    donor: „Great Britain“
                recipient: manufacturer1
                    theme: request1

request1:  request
                   target: „order“
                  speaker: „Great Britain“
                addressee:
                  message: „transport plane“
```

Should transfer "donor" role filler only if "speaker" is not already filled:

```
default_request_speaker(Request, Donor) :-
        receive(Receive),
        donor(Receive, Donor),
        theme(Receive, Request),
        request(Request).
```

Default Values

Insert default value as a role filler in absence of specific information

Example:

In Stock Market context use default "share" for "goods" role of "buy":

```
default_buy_goods(Buy, "share") :-
        'Buy is an event in a stock market context'.
```

Example:

Disjunctive (uncertain) information

Linguistic analysis is uncertain whether "Rover" or "Chrysler" was bought:

```
default_buy_goods(buy1,"Rover").
default_buy_goods(buy1,"Chrysler").
```

This amounts to *two* models, representing the uncertainty

They can be analyzed further

Default Value – General Transformation

Technique:

```
a :- not not_a.
not_a :- not a.
```

has two stable models: one where a is true and one where a is false

Choice to fill with default value or not:

```
goods(F,R) :-
        not not_goods(F,R),
        buy(F),
        default_buy_goods(F,R).

not_goods(F,R) :-
        not goods(F,R),
        buy(F),
        default_buy_goods(F,R).
```

Require at least one filler for role:

```
false :-
        buy(F),
        not some_buy_goods(F).
```

Case of waiving default value:

```
false :-
        buy(F),
        default_buy_goods(F,R1),
        goods(F,R1),
        goods(F,R2),
        not equal(R1,R2).

equal(X,X).
```

Role is filled:

```
some_buy_goods(F) :-
        buy(F),
        goods(F,R).
```

Conclusions

- Objective: "robust" reasoning support beyond description logics
- Method
  - FO theorem prover, specifically model generation paradigm
  - Tailor translation to capitalize on prover features
  - Exploit nonmonotonic features (for KB with FO semantics!)
- Practice
  - Experimental evaluation on OWL test suite "promising"
  - Need more experiments with e.g. OpenCyc and FrameNet