SKIL Studentenkonferenz Informatik Leipzig
25. September 2012

Relevanzbasiertes Preprocessing für automatische Theorembeweiser

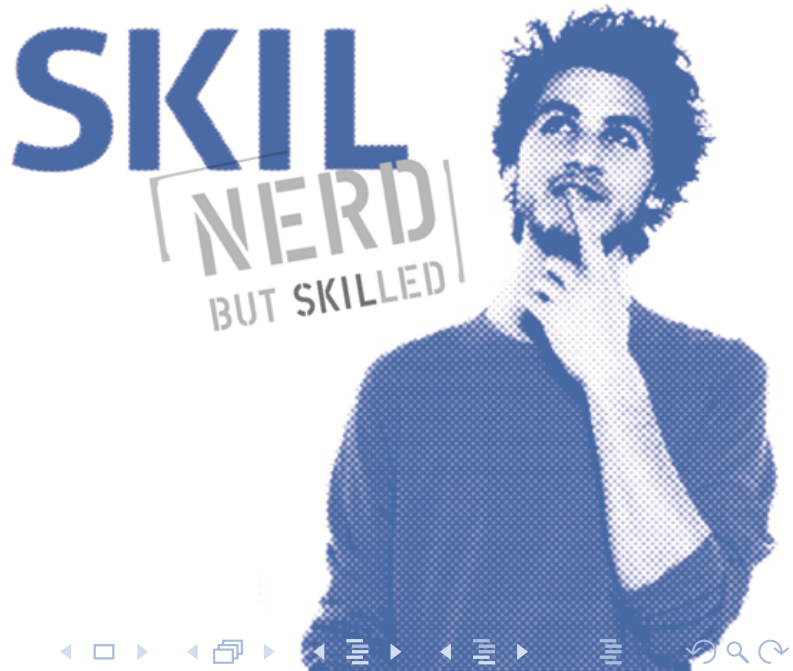
Mario Frank

Mario.Frank@Uni-Potsdam.de

25. September 2012

Agenda

Theorembeweiser



Anwendungsgebiete

Partielle Lösung der Fragen zur

- **Software-/Hardware-Verifikation**:
  Entspricht die Software/Hardware der Spezifikation? Hält eine konkrete Schleife?
- **Medizin**:
  Bei gegebener Menge von Symptomen: Welche Krankheiten kommen in Betracht?
- **Rechtswissenschaften**: Subsumtion
  Welche der Rechtsnormen sind auf den konkreten Sachverhalt anwendbar?

Grundlagen

Theorembeweiser suchen nach Beweisen für einen Satz mit einer gegebenen Menge von wahren Formeln (Axiome,Hypothesen, Lemmata,...).
Ansätze:

1. Beweis durch Widerlegung
   Annahme: Satz ist inkorrekt, zeige Unerfüllbarkeit
2. Beweis durch Konstruktion
   Annahme: Satz ist korrekt, suche Konstruktion

Grundlagen Kalküle

Ein logischer Kalkül ist ein Regelsatz für die

- Beweissuche (Analytischer Kalkül, z.B. Resolution, Tableau)
- Beweiskonstruktion (meist nach Analyse vorgenommen)

Neben dem Ansatz implementieren Theorembeweiser vor allem Kalküle. Manche Kalküle setzen Formeln in speziellen Normalformen, wie Negationsnormalform oder Konjunktive Normalform, voraus.

Grundlagen Konnektion

Eine Konnektion ist ein Paar von Prädikaten mit konträrer Polarität, die unifizierbar sind: $\{p, \neg p\}$.
Seien p_1 und p_2 Prädikate. Diese sind unifizierbar, wenn sie dieselbe Anzahl an Argumenten haben und es eine Substitution gibt, sodass sie gleich werden.

Beispiel - Tweety-Problem

$$
\begin{array}{l}
canFly(tweety) \\
\forall X((bird(X) \wedge \neg penguin(X)) \Rightarrow canFly(X)) \\
bird(tweety) \\
penguin(tux) \\
cat(meez)
\end{array}
$$

Abbildung: Prädikatenlogische Formelmenge - Tweety-Problem

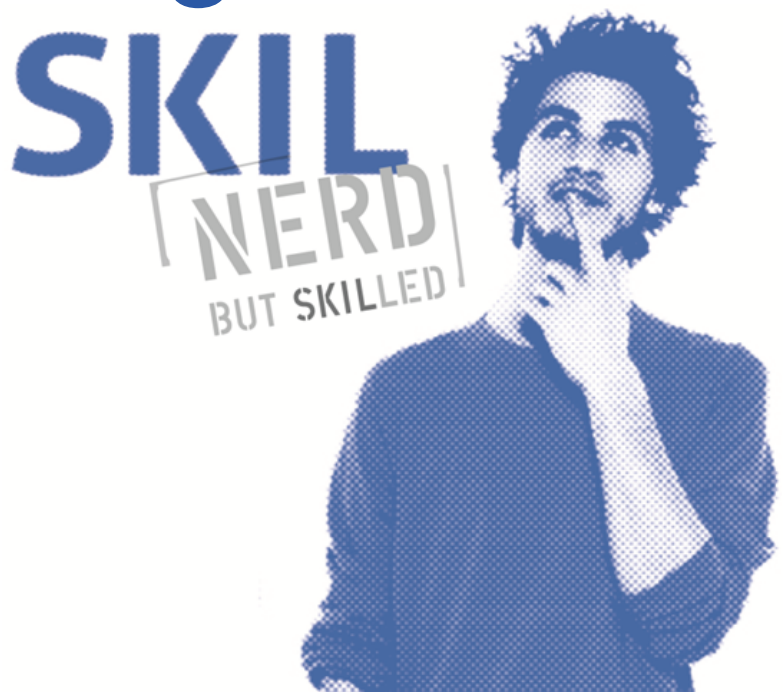
Grenzen der Theorembeweiser

Theorembeweiser können nur eine begrenzte Menge an Formeln in einer begrenzten Zeit behandeln. Viele interessante Probleme enthalten jedoch Tausende oder Hunderttausende Formeln.

Gründe für die Grenzen:

1. Speicherverbrauch der Formeln in Normalform
2. Rechenaufwand durch Beweissuche bei vielen Formeln (große Pfad-Anzahl)
3. Hoher Zeitaufwand für Normalform-Transformation
4. Nutzung von Interpretern (z.B. Prolog)

Überschreiten der Grenzen: Preprocessing



Preprocessing - Grundlagen

Ein Preprocessor soll die Anzahl der Axiome verringern und die Probleme (schneller) lösbar machen.

Ansätze:

- **Lexembasiert:** State ot the Art (z.B. SinE)
  Wähle Axiome, die Lexeme teilen. Lexeme sind hier Funktions-/Prädikats-/Konstantennamen.
- **Konnektions-basiert:** (z.B. ARDE)
  Wähle Axiome, die mit dem Problem über Konnektionen verbunden sind.

Beispiel - Tweety-Problem

$$
\begin{aligned}
&1) \neg canFly(tweety) \\
&2) \forall X((\neg bird(X) \vee penguin(X)) \vee canFly(X)) \\
&3) bird(tweety) \\
&4) penguin(tux) \\
&5) cat(meez)
\end{aligned}
$$

Abbildung: Skolem-Normalform - Tweety-Problem

Lexembasiert: 1,2,3,4
Konnektions-basiert:

- Iteration 1: 1,2
- Iteration 2: 1,2,3

Ansätze, beleuchtet

- **Lexembasiert:**
  1. Keine Normalform notwendig -> schnell
  2. Menge kann unvollständig werden
  3. Betrachtet nicht die Ersetzbarkeit von Variablen
- **Konnektions-basiert:**
  1. Braucht Negationsnormalform -> langsamer als lexembasiert
  2. Axiommenge in Iteration $\geq i$ vollständig
  3. Subsumiert Nutzung von Lexemen

ARDE - Konzept

Ablauf:

1. Transformiere alle Formeln in Skolem-Normalform
2. Suche für jedes Prädikat in der aktuellen Formelmenge eine Konnektion
3. Wähle jedes konnektierte Axiom, übergebe aktuell gewählte Menge und den Satz an den Beweiser
4. Weiter bei 2.

Folge: Es baut sich nach und nach ein Konnektionsgraph auf.

ARDE - technisches

- Implementiert in C++ (2500 loc.) und Prolog (200 loc.)
- Prädikats-Suche via Hash-Tabellen
- Struktur der Formeln wird nicht betrachtet (Konjunktionen/Disjunktionen gehen verloren)
- Originale der Formeln werden an den Beweiser übergeben -> **Jeder Beweiser kann ARDE nutzen**
- Nach jeder Iteration wird ein Beweiser-**Thread** gestartet

Stärken und Schwächen

- Gleichheiten werden (noch) nicht betrachtet -> teilweise unvollständige Mengen
- Struktur der Formeln wird unzureichend betrachtet -> noch teilweise große Mengen
- Lernt Konnektionen und kann diese später wiederverwenden
- Kann mehrere Hunderttausend Formeln verarbeiten
- Arbeitet zielorientiert

Tests von leanCoP und E auf aktueller ARDE-Version

Die Wettbewerbs-Probleme wurden nochmals bearbeitet und die Ausgaben von ARDE an leanCoP 2.2 und E 1.6 (ohne SinE) übergeben.

Klasse	Axiome
ISA	50-6.000
MZR	30-50.000
SMO	30.000 - 7.000.000

Tabelle: Problemklassen - übliche Axiomanzahl

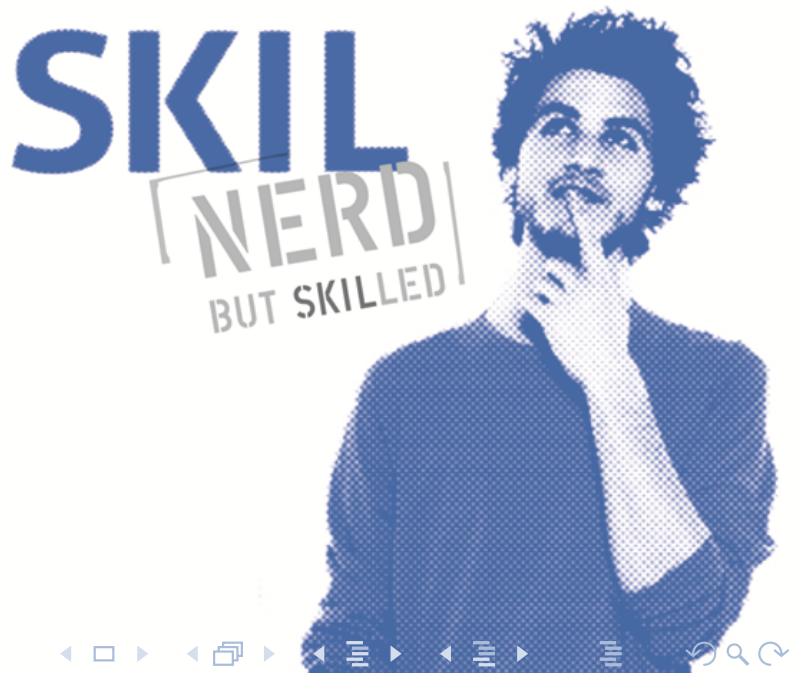
Verbesserungen durch ARDE

Klasse	Axiome	Auswahl	E	leanCoP
ISA (75)	50-6.000	14-4.500	+0	+4
MZR (80)	30-50.000	< 10.000	+3	+1
SMO (20)	ca. 55.000	< 5.000	+6	+3

Tabelle: Anzahl der zusätzlich lösbaren Probleme

SMO-Probleme konnten in 60 Sekunden ohne Preprocessing nicht gelöst werden.
Die 3 MZR-Probleme konnten von E (mit SinE) im Wettbewerb nicht gelöst werden.
Preprocessing-Dauer: meist < 5 sec.

Fazit

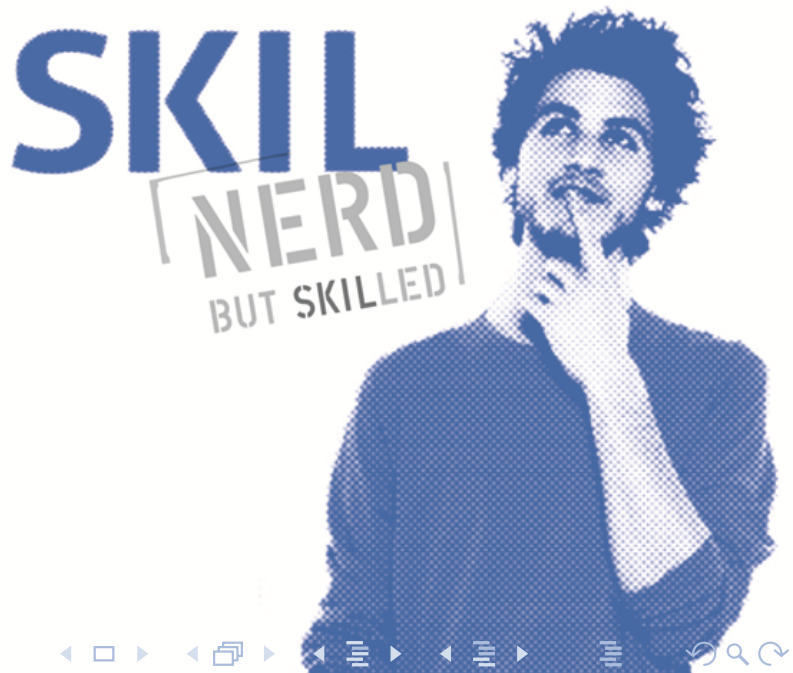


Fazit / Weitere Entwicklung

Das Grundkonzept funktioniert, ist aber noch zu schwach.
Kommende Features sind:

- Behandlung von Gleichheiten
- Nutzung der Formelstruktur
- Anpassung an Modallogik und intuitionistische Logik
- Stärkere Parallelisierung

Ende



Vielen Dank für Ihre Aufmerksamkeit