Lehrstuhl für Softwaretechnik und Programmiersprachen
Prof. Dr. Michael Leuschel
Joshua Schmidt
Alexandros Efremidis

Heinrich Heine Universität Düsseldorf

Einführung in die logische Programmierung – Wintersemester 2019
Programmierprojekt

1 Abgabe und Bewertung

Für das Erreichen der Klausurzulassung genügt es wenn Ihre Abgabe alle Tests und Benchmarks sinnvoll besteht, unabhängig von der Laufzeit.

Das Projekt ist in **Einzelarbeit** zu bearbeiten. Sie können sich natürlich untereinander austauschen, aber bitte keinen Code teilen.

Letzter Abgabetermin: 21.01.2020

Wir möchten einen kleinen Wettbewerb um die schnellsten Solver starten. Ihre finale Version des SAT solvers sollen Sie bitte per E-Mail an joshua.schmidt@hhu.de einreichen. Mehrfache Abgaben sind hierbei erlaubt. Nach jeder Abgabe wird die Laufzeit auf der Homepage aktualisiert. Es zählt die Performance der letzten Abgabe. Es ist also ratsam vor dem Abschicken Benchmarks zu machen.

Bis zum Abgabetermin lassen wir die abgegebenen Solver um die Wette rechnen.

Bei Problemen oder Fragen melden Sie sich bitte rechtzeitig.

2 Aufgabe: SAT Solver

Ziel ist es ein Prolog Programm zu implementieren, dass Formeln in Aussagenlogik in konjunktive Normalform umformen und dann lösen kann. Verwenden Sie hierfür die bereitgestellte Datei `sat_solver.pl`.

Die Eingabe erfolgt als Syntaxbaum. Dieser setzt sich rekursiv zusammen aus

- Literalen $lit(\ldots)$, also
  - Den Konstanten `lit(true)` und `lit(false)`,
  - Prolog Variablen als Platzhaltern, also z.B. `lit(X)`,
- Der Implikation $\rho \Rightarrow \phi$, als Prolog Term `implies(`ρ`,`ϕ`)`,
- Der Konjunktion $\rho \wedge \phi$, als Prolog Term `and(`ρ`,`ϕ`)`,
- Der Disjunktion $\rho \vee \phi$, als Prolog Term `or(`ρ`,`ϕ`)`, und
- Der Negation $\neg\rho$, als Prolog Term `not(`ρ`)`.

Die Terme können beliebig geschachtelt auftreten.

Ihr SAT Solver soll drei Prädikate zur Verfügung stellen:

- `solvername/1`. Es soll ein Prolog Atom mit dem Namen Ihres Solvers zurück gegeben werden, welches für den Wettbewerb verwendet wird. Z.B. `solvername(prolog_rules).`

- `to_cnf/2`. Das erste Argument ist eine Aussagenlogische Formel in der oben beschriebenen Form. Die konjunktive Normalform soll als Liste von Listen im zweiten Argument zurückgegeben werden. Dabei ist jede innere Liste eine Klausel (also Disjunktion), die Listen untereinander werden mit $\wedge$ verbunden.

  Beispiele: `and(lit(X),lit(true))` wird umgeformt zu `[[X],[true]]`,
  `and(lit(X), or(lit(true),lit(false))` wird umgeformt zu `[[X],[true,false]]`.

  Negation wird weiterhin durch *not*(...) ausgedrückt:
  `and(not(lit(X)),lit(true))` wird umgeformt zu `[[not(X)],[true]]`.

- `solve/1`. Das Prädikat bekommt eine CNF im oben beschriebenen Format und versucht eine Variablenbelegung zu finden mit der die Aussagenlogische Formal wahr ist. Die Variablen (im Beispiel `X`) sollen entsprechend der Lösung mit `true` oder `false` unifiziert werden. Implementieren Sie hierfür den in der Vorlesung besprochenen **DPLL Algorithmus**.

Die Benchmarks 1-7 in dem Ordner `resources/sat_benchmarks/` sind genau die, welche für den Wettbewerb verwendet werden. Ihr Solver soll diese Formeln lösen können. Die Datei `large_formula_example.cnf` beinhaltet eine komplexere Formel, welche unsere SAT solver mit dem reinen DPLL Algorithmus vermutlich nicht in akzeptabler Zeit lösen kann. Diesbezüglich sind weitere (zum Teil nicht offensichtliche) Optimierungen erforderlich wie z.B. *conflict-driven clause learning*, *backjumping*, *two-watched literals* oder *random restarts*. Dies übersteigt aber den Rahmen dieser Einführungsveranstaltung und ist unter anderem Teil des Kurses “Vertiefung in die logische Programmierung”, welcher z.B. im Sommersemester 2020 angeboten wird.