Kapitel 1 – Einführung & Aussagenlogik

- Syntax: Wie werden Formeln gebildet.
- Semantik: Was ist die Bedeutung einer Formel.
- Kalkül: Wie kann die Gültigkeit einer Formel berechnet werden?
- Reihenfolge der Priorität (abnehmend): $\neg, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow$
- Interpretation mithilfe einer Belegung *P (v : F → B)*

$$\nu(x) = \nu_0(x) \text{ für } x \in \mathcal{P}$$
$$\nu(\top) = 1$$
$$\nu(\bot) = 0$$
$$\nu(\neg P) = \texttt{if } \nu(P) = 1 \texttt{ then 0 else 1}$$
$$\nu(P \vee Q) = \texttt{if } \nu(P) = 0 \texttt{ then } \nu(Q) \texttt{ else 1}$$
$$\nu(P \wedge Q) = \nu(\neg(\neg P \vee \neg Q))$$
$$\nu(P \rightarrow Q) = \nu(\neg P \vee Q)$$
$$\nu(P \leftrightarrow Q) = \nu((P \rightarrow Q) \wedge (Q \rightarrow P))$$
$$\nu(P \oplus Q) = \nu(\neg(P \leftrightarrow Q))$$

- **Erfüllbarkeit**: gibt Belegung für True
- **Kontingenz**: gibt Belegung für True und für False
- **Tautologie**: jede Belegung ist True
- **Kontradiktion/Unerfüllbar**: jede Belegung gibt False
- **Semantische Äquivalenz**: $P \equiv Q$ äquivalent zu $\vDash P \leftrightarrow Q$
- **Semantische Folgerung**: $P \vDash Q$ äquivalent zu $\vDash P \rightarrow Q$
- Äquivalenzumformungen: Distributivgesetze, Absorptionsgesetze, Kommutativität, Assoziativität, Doppelte Negation, DeMorgansche Gesetze
- SAT: Ist eine gegebene Formel P erfüllbar oder nicht.
- Alle state-of-the-art-Solver benützen CNF (Konjunktive Normalform)
- **Erfüllbarkeitsäquivalenz**: Zwei Formeln P und Q sind erfüllbarkeitsäqivalent, falls gilt: P ist erfüllbar genau dann, wenn Q erfüllbar ist. (schwächer als Äquivalenz)
  $x \rightarrow y$ und $z \wedge u \vee v$ sind erfüllbarkeitsäquivalent
- **Tseitin-Plaisted-Greenbaum**: Ein Verfahren zum Erstellen der CNF
  Ersetze $P = A \vee (B \wedge C)$ zu $P' = (A \vee x) \wedge (\neg x \vee N) \wedge (\neg x \vee C)$
  Vorteil: keine Verdopplung von A, keine exponentielles Wachstum
  Nachteil: Es kann mehr erfüllende Belegungen geben als im Ursprüngliche P
- Üblicherweiße Darstellung im **DIMACS**-Format (Jede Zeile endet mit einer 0)

Kapitel 2 - Der DPLL Algorithmus

- Grundidee: Fallunterscheidung und Vereinfachungen (UP)
- **Subsumption**: Klausel C subsumiert Klausel D
  $C \subseteq D$ ➔ $C \vDash D$ ➔ $\{C, D\} \equiv \{C\}$
- **Resolution**: $C \cup \{\neg x\}$ und $D \cup \{x\}$ ➔ $C \cup D$
- **Unit Resolution**: $C \cup \{\neg x\}$ und $\{x\}$ ➔ C
- Empty Clause ($\nu(P) = 0$): Mit aktueller Belegung wertet die Klausel zu 0 aus. ➔gesamte Formel P damit nicht erfüllbar
- **Worst-Case**: Alle 2^n Belegungen müssen geprüft werden
  ➔ tritt in der Praxis quasi nie auf (Heuristiken, …)

Input: Formel P in Aussagenlogik, Interpretation ν_0
Output: 1 oder 0
$DPLL(P, \nu_0)$

UnitPropagation(P, ν_0);
if $(\nu(P) = 1)$ **then**
  return 1
if $(\nu(P) = 0)$ **then**
  return 0
wähle eine noch nicht belegte Variable $x \in var(P)$
if $DPLL(P \cup \{\{\neg x\}\}, \nu_0 \cup [x \mapsto 0])$ **then**
  return 1
else
  return $DPLL(P \cup \{\{x\}\}, \nu_0 \cup [x \mapsto 1])$

Heuristiken

- numpos(x): Anzahl der positiven Vorkommen der Variable x in unerfüllten Klauseln
- numneg(x): Anzahl der negativen Vorkommen der Variable x in unerfüllten Klauseln
- **DLCS** (Dynamic Largest Combined Sum): Wähle als nächste Variable x mit der größten Summe: numpos(x) +ü numneg(x) (Idee: häufig vorkommende Variablen haben großen Einfluss)
- **DLIS** (Dynamic Largest Individual Sum): Wähle als nächste Variable x mit dem größten numpos(x) oder numneg(x) (Idee: häufig vorkommende Variablen haben großen Einfluss)
- **MOM** (Maximum occurence in clauses of minimal size): Wähle die Variable, deren Vorkommen in kurzen Klauseln maximal ist. (Idee: kurze Klauseln haben mehr Aussagekraft)
- **Aktivitätsheuristiken**: Wähle die Variable, die am aktivsten ist.

Kapitel 3 – CDCL SAT Solving

- **C**onflict **D**riven **C**lause **L**earning
- Idee: Lerne im Konfliktfall eine zusätzliche Klausel. Nicht-chronologisches Backtracking
- Typen von Klauseln die man lernen kann:
  - Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*
  - 1st new clause: Aus Resolution der Konflikte
  - **1UIP** (First Unique Implication Point): Solange resolvieren bis die Klausel Unit ist.
  - decision clause: enthält nur noch Entscheidungsvariablen
- Klauseln werden durch Resolution von Reason und Conflict Clause erstellt.

```
Input: Clauseset C
Output: SAT or UNSAT
level = 0 ;
α = ∅ ;
while true do
    UP(C, α) ;
    if C contains an empty clause then
        ec = empty clause;
        level = analyzeConf(ec, C) ;
        if level == −1 then
            return UNSAT
        backtrack(level) ;
    else
        if α ⊨ C then
            return SAT
        level = level + 1 ;
        choose x ∉ α ;
        α = α ∪ [x ↦ 0] ;
```

- Der Algorithmus gibt nur dann SAT zurück, wenn erfüllende Belegung α gefunden wurde. Partielle Korrektheit + Termination ➔ Totale Korrektheit

Kapitel 4 – CDCL SAT-Solving Zutaten

- Klauselllernen
- Nicht-Chronologisches Backtracking
- Gute Auswahlheuristik (z.B. VSIDS (variable state independent decaying sum))
- Effiziente Unit Propagation (watched literals)
- Restarts mit Clause Deletions (lokale Minima vermeiden, 1. Restart nach 100 Klauseln, 2. Restart nach 150, 3. Restart nach 300 Klauseln)
- ➔ zu finden in MiniSAT (< 1000 Zeilen Code)

Kapitel 5 – Implementierung effizienter SAT Solver

Aktivitätsheuristik (VSIDS):

- Jede Variable bekommt eine Aktivität zugewiesen. Initial ist die Aktivität gleich der Anzahl der Vorkommen in den Klauseln.
- Für jede gelernte Klausel erhöhe die vorkommenden Variablen
- Teile periodisch alle Aktivitäten durch einen Faktor
- Wähle stets die Variable mit höchster Aktivität.

Effiziente Unit Propagation:

- SAT Solver befinden sich ca. 90% der Zeit in Unit Propagation
- Durch UP wird der exponentiell wachsende Lösungsraum eingeschränkt
- **Idee**: Es genügt die Information ob Klausel Unit oder Leer ist
- Es genügt 2 Literale pro Klausel zu beobachten ➔ Watched Literals
- *Vorteil*: Die Anzahl der Referenzen pro Klausel ist konstant 2 (nach Backtracking können die aktuellen Referenzen beibehalten werden)
  Nachteil: Komplette Klausel muss geprüft werden um zu wissen ob UNIT oder UNSAT

Verbessertes Lernen:

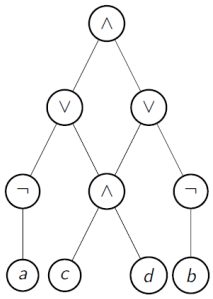
- **Self-Subsuming Resolution**: Resolution zwischen zwei Klauseln $A \cup \lambda$ und $B \cup \{\neg\lambda\}$ wobei $A \subset B$. Die Resolvente ist dann B und subsumiert die Klausel $B \cup \{\neg\ \lambda\}$.
- **Lokale Minimierung**: Berechne die 1UIP Klausel. Führe Self-Subsuming Resolution in umgekehrter Belegungsreihenfolge der Literale aus (Resolution jeweils mit den Reasons der Belegung)
- **Rekursive Minimierung**: Ein Literal kann aus der 1UIP Klausel u entfernt werden, wenn alle Literals seiner Reason im Implikationsgraph von Literalen aus u dominiert werden.

Clause Deletion:

- N-order learning: Nur Klauseln mit n oder weniger Literalen werden gelernt.
- M-size relevanze-based learning: Klauseln warden nur temporär gelernt.
- K-bounded learning: Klauseln mit weniger als k Variablen werden gespeichert. Größere Klauseln werden gelöscht sobald sie nicht mehr unit sind.

Restart Strategien:

- Anzahl der Konflikte nach denen neu gestartet wird, wird mit jedem Restart erhöht.
- **Luby Sequenz**: 20, 40, 20, 40, 80, 20, 40, 80, 160, …



Kapitel 6 – Non-CNF SAT-Solving

- *Problem*: Anwendungsprobleme liegen zunächst nicht in Normalform (CNF) vor
- *Idee*: SAT-Solving auf nicht-normalisierten Formeln
- Klauseldarstellung nicht mehr möglich ➔ DAT-Darstellung
- Spezielle Regeln für innere Knoten ∧∨ ¬
- Integration moderner Techniken: CDCL, watched Literals
- **Don't case-Propagation**: Hängt der Wert eines Knotens nicht mehr vom Wert einer der Teilformeln ab, so kann der Teilformel der Wert * (don't care) zugewiesen werden. (Implementierung: Watched-Literals-Schema)
- Grundidee des Konfliktlernens: Ausgehende von beiden Belegungen der Konfliktvariable die Belegungen zurückverfolgen. Die gelernte Klausel ist die Disjunktion der negierten Variablen aus der NoGood-Menge. ➔ 1UIP-Lernen
- Gründe für Belegungen merken:
  - Decision: Aufgrund von gewählter Belegung
  - Parent: Setzen der Variable aufgrund des Wertes eines Elternknotens
  - Child: Setzen der Variable aufgrund des Wertes eines Kindknotens
  - NoGood: Setzen der Variable aufgrund eines gelernten NoGood

Zusammenfassung: Keine CNF-Konversion erforderlich, enthält Formelstruktur, schnelle BCP und CDCL möglich, komplizierte Implementierung, kompliziertere Datenstruktur

Kapitel 7 – QBF Solving

```
level := 0;
while true do
    unitPropgation();
    if a conflict is reached then
        level := analyseConflict();
        if level = 0 then
            return false
        backtrack(level);
    else
        if formula is satisfied then
            level := analyseSAT();
            if level = 0 then
                return true
            backtrack(level)
        else
            level := level + 1;
            choose an unassigned x ∈ var(P)
            (wrt. the q-level);
            α := α ∪ [x ↦ 0];
```

- QBF-Solving ist PSPACE-vollständig
- Umwandlung in Pränexe Normal Form (PNF) notwendig
  $Q_1x_1, \dots, Q_nx_n\psi$ dabei ist ψ quantorenfrei
- Zusätzliche Quantifikationslevel (QL) (pro Quantorenwechsel)
  Variablen von außen nach innen wählen. Innerhalb QL wählbar.
  UP auch über QL hinweg.
- $E(C)$ Literale mit existenzquantifizierten Variablen
- $U(C)$ Literale mit allquantifizierten Variablen
- $ql(x)$ Quantifikationslevel einer Variable x
- **Empty Clause**: Eine Klausel C ist unerfüllbar, wenn
  - $\forall e \in E(C)$ gilt $v(e) = \bot$
  - $\forall u \in U(C)$ gilt $v(u) = \top$
- **Unit Clause**: Eine Klausel C ist unit, wenn
  - $\exists e \in E(C): v(e) = nil$
  - $\forall e \in E(C): e' \neq e \wedge v(e') = \bot$
  - $\forall u \in U(C): v(u) \neq \top$
  - $\forall u \in U(C): v(u) = nil \rightarrow ql(u) > ql(e)$
- Allquantifizierte Variablen erhalten ein Flag, damit beide Belegungen getestet werden
- Formatierung im QDIMACS Format (Solver: Quaffle, Quantor, QuBE, SKizzo)
- Erweiterungen auf PSAT (parametric SAT) und PQSAT (parametric QSAT)
  Lösungsansatz PQSAT: Top-Level DPLL und an jedem Blatt SAT/QBF
  Lösungsansatz PSAT: Clause Distribution, Model Enumeration, DNNF

Kapitel 9 – MaxSAT und Pseudo Boolesche Optimierung

- MaxSAT: Finde eine Belegung, welche die Anzahl erfüllter Klauseln maximiert.
- Partial MaxSAT: Es gibt harte Klauseln die erfüllt sein müssen.
- Weighted MaxSAT: Alle Klauseln sind weich, aber mit verschiedenen Gewichten versehen.
  Notation: (c, w) bei harten Klauseln $w = \infty$
- **Branch & Bound Algorithmus ➔**
  Berechnet die minimalen Kosten

```
Input: MaxSAT Instanz φ, Obergrenze UB
φ = simplifyFormula(φ)
if φ = ∅ oder φ enthält nur leere Klauseln then
    return #emptyClauses(φ)
LB = #emptyClauses(φ) + underestimation(φ)
if LB ≥ UB then
    return UB
x = selectVariable(φ)
UB = min(UB, BnB(φ_x, UB))
return min(UB, BnB(φ_x̄, UB))
```

- MaxSAT lösen mit Hilfe von SAT Solving
  Vorteil: Verwendung effizienter Techniken aus SAT
  Idee: Rufe iterativ SAT Solver auf bis Optimum gefunden ist
- SAT-basierter Ansatz von Le Berre: Liefert nicht zwangsläufig immer den kleinsten UNSAT core
- **Fu & Malik Algorithmus ➔**
  Idee: Prüfe iterativ mit Hilfe eines SAT Solvers (mit unsat core), ob Instanz erfüllbar ist.
- **PM2 Algorithmus**: Optimierung von Fu & Malik Algorithmus. Jede Klausel erhält nur eine blocking Variable. Limitiere Erfüllbarkeit der blocking Variablen durch cardinality constraint.

```
Input: Partial MaxSAT Instanz φ
cost ← 0
while true do
    (st, φ_c) ← SAT(φ)
    if st = SAT then return cost
    BV ← ∅
    foreach C ∈ φ_c do
        if C is soft then
            b ← new blocking variable
            φ ← φ \ {C} ∪ {C ∨ b}
            BV ← BV ∪ {b}
    if BV = ∅ then return None
    φ ← φ ∪ CNF(∑_{b∈BV} b = 1)
    cost ← cost + 1
```

- **WPM1 Algorithmus**: Erweiterung von Fu & Malik Algorithmus zu Partial Weighted MaxSAT.

Kapitel 10 – Software Verifikation mit Bounded Model Checking

- **Softwareverifikation**: Beweis der Korrektheit eines Programms mit mathematischen Methoden
  Grundsätzliche Einschränkung: Es kann nur verifiziert werden, was zuvor spezifiziert wurde. Bei falscher Spezifikation bringt Verifikation nichts. …
  Universelle Spezifikationen: keine Nullpointer Dereferenzierung, kein Arrayüberlauf, nur positive Werte in unsigned Variablen, …
- Verschiedene Verfahren zur Softwareverifikation
  - Software Bounded Model Checking (SBMC) (Dicht am Quellcode, weitgehend automatisiert, aber begrenzte Schleifendurchläufe/Rekursionen)
  - Theorembeweiser (Ausdrucksstark, bedarf menschlicher Begleitung)
  - Hoare-Logik mit Theorembeweiser
  - Konventionelle Statische Codeanalyse

Ein formales Modell für Software

- Formalisierung eines Programms als Transition System
- Zustand speichert: Stand des Program Counter, Variablenbelegungen, Ausführungsstack
- Gegenbeispiel: Können bestimmte „schlechte“ Programmzustände (error locations) erreicht werden? (Reachability Problem)

Bit Vektoren

- Ein Bit-Vektor der Länge n ist eine Sequenz von n Boolschen Variablen. b[i] notiert das i-te Bit von Vektor b (von links, beginnend mit 0).
- Bit Blasting: Eingabeformel wird in Bit-Vektor Arithmetik umgewandelt.
  Beispiel: Auas 32 Bit integer wird eine Bit-Vektor

Bounded Model Checking

- Zur Übersetzung eines Programms in die Bitvektorgleichung sind 5 Schritte notwendig:
  - Syntaktischen Zucker der Programmiersprache entfernen
  - Schleifen k-mal ausrollen
  - Programm in Single Static Assignment Form transferieren
  - Bitvektorgleichung für einzelne Zuweisungen erstellen
  - Übersetzen der Bitvektorgleichung mit Hilfe von Bit-Blasting in Aussagenlogische Formel

Kapitel 11 – SAT Modulo Theories (SMT Solving)

- Codierung nach SAT zwar oft problemlos möglich, aber Codierungen werden oft zu groß
- Für viele Theorien sind bereits Entscheidungsverfahren bekannt, die mehr Domänenwissen einbeziehen, als eine Codierung nach SAT
  (Gleichheitslogik, uninterpretierte Funktionen, lineare Arithmetik, Arrays, Zeigerlogik, …)
- Kombinierte Theorien: Wenn zwei Theorien T_1 und T_2 in einer Formel kombiniert werden, besitzen beide Signaturen von T_1 und T_2 gemeinsame Symbole, welche eventl. nicht dieselbe Bedeutung (selbe Relation bzgl. Des Universums) haben. Eines muss umbenannt werden.
  ➔ Theorie ist (bei uns) eine Menge von Modellen
  Die Kombination $T_1 \oplus T_2$ von T_1 und T_2 ist die Menge aller Modelle B, deren Reduktum_1 isomorph ist zu einem Modell von T_1 und deren Reduktum_2 isomorph zu einem Modell von T_2.
- Assoziiere mit jeder Signatur Σ eine Signatur Ω, die enthält aussagenlogische Konstanten von Σ und die Menge an neuen aussagenlogischen Symbolen mit derselben Kardinalität.

- **Kongruenzhüllen Algorithmus**:
  Initialisierung – Vereinigen – Kongruenzhülle bilden

Ansätze zum Entscheiden von SMT Problemen

- **Eager Approach**: Übersetze gesamtes Problem in eine erfüllbarkeitsäquivalente Formel in Aussagenlogik und benutze SAT Solver.
- **Lazy Approach**: Rufe speziellen Theory Solver auf, wenn er gebraucht wird.
  Was sollte ein Theorie T-Solver können? Model Generation, Conflict Set Generation, Incrementality, Backtrackability, Deduction of Unassigned Literals and Interface Equalities
- Offline Integration: einfach, aber SAT wird immer wieder neu gestartet
- Online Integration: intelligentere Integrationsform, Backtracking und Lernen des DPLL-Solvers kann vom T-Solver profitieren, aber T-DPLL profitiert nicht automatisch von Verbesserungen im SAT Solving
- Das ***T*-DPLL Framework ➔**

Input: $\mathcal{T}$-Formel φ, $\mathcal{T}$-Belegung Γ
Output: SAT oder UNSAT
if $\mathcal{T}$−preprocess(φ, Γ) == *Conflict* **then**
  return *UNSAT*
while *true* **do**
  $\mathcal{T}$−decide_next_branch(φ^P, Γ^P)
  while *true* **do**
    status = $\mathcal{T}$−deduce(φ^P, Γ^P)
    if *status* == $\mathcal{T}$-*SAT* **then**
      $\Gamma = \mathcal{B}2\mathcal{T}(\Gamma^P)$
      return *SAT*
    else if *status* == $\mathcal{T}$-*Conflict* **then**
      blevel = $\mathcal{T}$−analyze_conflict(φ^P, Γ^P)
      if *blevel* == *-1* **then**
        return *UNSAT*
      $\mathcal{T}$−backtrack$(blevel, \varphi^P, \Gamma^P)$
    else
      break

Kombination von Theorien

- **Nelson-Oppen Methode**: Eigener Solver für jede Theorie und die Solver können „Interface"-Informationen untereinander austauschen.
- Die Theorien müssen die folgenden Eigenschaften erfüllen:
  - T_1, …, T_n sind quantorenfreie first-order Theorien mit Gleichheit
  - Es gibt jeweils eine Entscheidungsprozedur für T_1,…,T_n
  - Die Signaturen sind disjunkt, d.h. für alle
    $$1 \leq i < j \leq n, \Sigma_i \cup \Sigma_j = \emptyset$$
  - T_1,…, T_n werden über unendlichen Domänen interpretiert (z.B. lineare Arithmetik über R, aber nicht Theorie der endlichen breiten Bit Vektoren)
- Die Methode wird wesentlich effizienter wenn zusätzlich gilt, dass die Theorien auch konvex sind.
- **Konvexe Theorie**: Wenn eine Formel eine Disjunktion von Gleichungen impliziert, impliziert sie mindestens eine dieser Gleichungen separat.
- Durch **Purification** werden die Theorien aufgeteilt und erhalten Interfacevariablen zur angrenzenden Theorie.
- Bei nicht konvexen Theorien ist ein Case-Split notwendig als 4ter Schritt.

- **Eingabe:** Konjunktion φ über verschiedenen konvexen Theorien $T_1, \ldots, T_n$
- **Ausgabe:** SAT, wenn φ erfüllbar ist, UNSAT sonst

1. **Purification:** Purifizieren von φ zu $\varphi' = \{F_1, \ldots, F_n\}$ mit $F_i \in T_i$.
2. T_i-**Decision:** Wende Entscheidungsverfahren für T_i auf F_i an
   - Wenn ein i existiert, so dass F_i in T_i nicht erfüllbar ist, Rückgabe: UNSAT
3. **Equality Propagation:** Wenn i und j existieren, so dass
   - F_i in T_i eine „Interface"-Gleichung $a = b$ mit zwischen T_i und T_j geteilten Hilfsvariablen impliziert und
   - diese Gleichung aber noch nicht von F_j in T_j impliziert wird,

   dann füge diese Gleichung zu F_j hinzu und gehe wieder zu Schritt 2.
4. Rückgabe SAT

Beispiel (Case Split bei nicht-konvexen Theorien)

- $\varphi' = (1 \leq x) \wedge (x \leq 2) \wedge p(a_0) \wedge \neg p(a_1) \wedge \neg p(a_2) \wedge a_0 = x \wedge a_1 = 1 \wedge a_2 = 2$
- F_1 impliziert $x = 1 \vee x = 2 \Rightarrow$ **Case Split**

F_1 (LA über $\mathbb{Z}$)	F_2 (EUP)
$1 \leq x$	$p(a_0)$
$x \leq 2$	$\neg p(a_1)$
$a_0 = x$	$\neg p(a_2)$
$a_1 = 1$	
$a_2 = 2$	
$x = 1$	
$x = a_1$	
$a_0 = a_1$	$a_0 = a_1$
	false

F_1 (LA über $\mathbb{Z}$)	F_2 (EUP)
$1 \leq x$	$p(a_0)$
$x \leq 2$	$\neg p(a_1)$
$a_0 = x$	$\neg p(a_2)$
$a_1 = 1$	
$a_2 = 2$	
$x = 2$	
$x = a_2$	
$a_0 = a_2$	$a_0 = a_2$
	false

- $a_0 = a_1$ bzw. $a_0 = a_2$ sind die einzigen abgeleiteten Gleichungen in der Theorie LA über $\mathbb{Z}$, die auch in EUP sind und deshalb propagiert werden.
- In beiden Fällen ist die Rückgabe false, also ist Gesamtergebnis UNSAT