Technische Universität Braunschweig

Algorithmen und Datenstrukturen 2 – Übung #4

Phillip Keldenich
14.06.2023

Heute: Backtracking

- Entscheidungsprobleme
- Backtracking & Propagation: Beispiel Sudoku
- Probleme von Backtracking
- Boolesches Erfüllbarkeitsproblem (Satisfiability, SAT)
- Exkurs: CDCL (Conflict-Driven Clause Learning) für SAT

Entscheidungsprobleme

Wir haben in der Vorlesung oft Optimierungsprobleme betrachtet.

Manchmal reicht es aber, Entscheidungsprobleme zu betrachten.

- Beispielsweise eine Knapsack-Eingabe und ein Schwellwert c.
- Als Ausgabe ist nur eine Antwort ‘Ja’ oder ‘Nein’ gefragt:
  - Gibt es eine Möglichkeit, Objekte im Wert von mindestens c mitzunehmen?
  - (Falls ‘Ja’, will man oft auch eine mögliche Lösung haben.)
- Anderes Beispiel: Sudoku
  - Gegeben teilweise ausgefülltes Sudoku, ist es möglich das Feld komplett auszufüllen?
- Wie lösen wir schwere Entscheidungsprobleme?

Problemmodellierung & mögliche Lösungen

Angenommen, unser Problem lässt sich wie folgt darstellen:

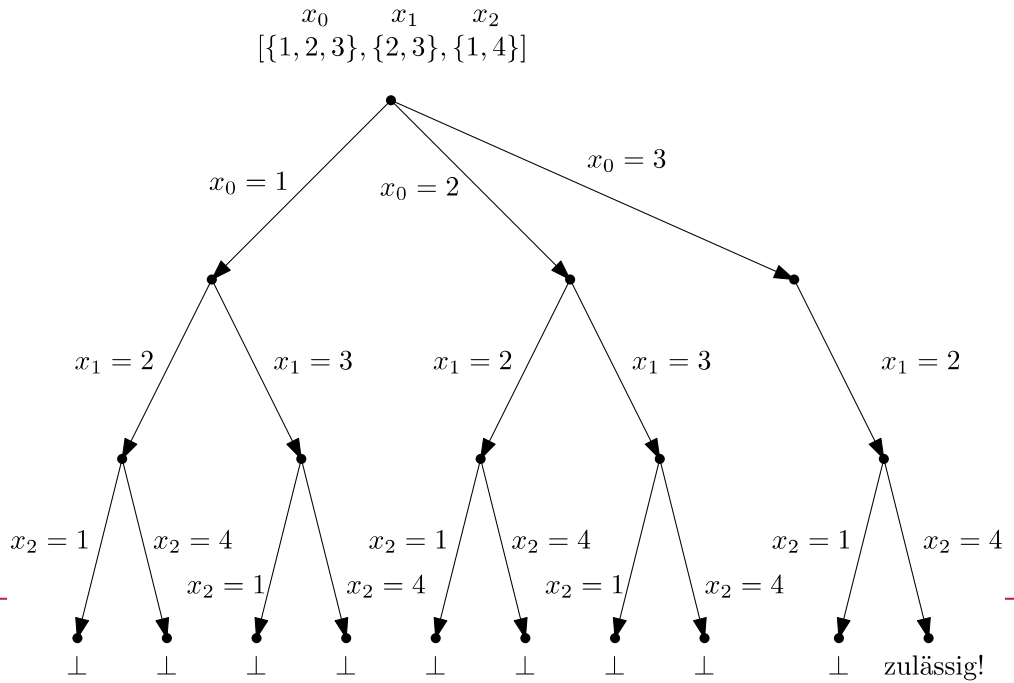
- Wir haben eine endliche Anzahl Variablen $x_1, \dots, x_n$ mit endlich vielen möglichen Werten
- Wir haben Nebenbedingungen (Einschränkungen) an die Belegungen
- Z.B. Knapsack mit Mindestgewinn c:
  - Variablen $b_1, \dots, b_n \in \{0,1\}$
  - Kapazitätsbedingung: $\sum_{k=1}^{n} b_k z_k \leq Z$
  - Gewinnbedingung: $\sum_{k=1}^{n} b_k p_k \geq c$
- Z.B. Sudoku:
  - Eine Variable $x_f \in \{1, \dots, 9\}$ pro Feld
  - Für jede Gruppe (Zeile/Spalte/Block) G und jedes $i \in \{1, \dots, 9\}$:
    - $\bigvee_{f \in G} x_f = i$
    - $\forall f, g \in G,\ f \neq g\colon\ x_f \neq x_g$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Suchbaum/Enumerationsbaum

Wiederholung: Suchbäume wie bei vollständiger Enumeration / Branch & Bound

Algorithmus B_1
Voraussetzung: Variablen $x_0, \ldots, x_{n-1}$, Nebenbedingungen
Parameter: Array von Mengen erlaubter Werte $\mathcal{D} = [D(x_0), \ldots, D(x_{n-1})]$
Rückgabe: zulässige Lösung oder $\perp$



Pruning (Suchbaum beschneiden)

Welche Teile des Baums können wir abschneiden?

Was können wir an einem inneren Knoten v (Teilbelegung!) tun?

Zulässigkeitstest (problemabhängig!):

Wenn wir wissen, dass kein Knoten unter v zulässig ist, können wir abschneiden!

Test muss korrekt sein (aber nicht vollständig, *false positives* sind erlaubt).

Beispiel: Zulässigkeitstest bei Knapsack mit Mindestgewinn c?

- Wie bei Branch & Bound: Gewählte Objekte überschreiten Gewicht $Z \rightarrow$ unzulässig
- Was geht noch?
- Analog zu Pruning bei Branch & Bound: $UB(v) < c \rightarrow$ unzulässig
- Damit erhalten wir einen sehr ähnlichen Algorithmus zu Branch & Bound!

Propagation

Weitere Verbesserung des Algorithmus:

- Oft gibt uns eine fixierte Teilbelegung weitere Teile der Belegung vor
- Beispielsweise bei Sudoku:
  - Die meisten Menschen lösen die meisten Sudoku ohne ‘ausprobieren’
  - Sie haben also in der Regel einen Suchbaum mit nur einem Knoten
  - Weil die initiale Teilbelegung durch logische Folgerungen erweitert werden kann!
  - Dies nennt man ‘Propagation’ (‘Fortsetzung/Verbreitung’ bisheriger Entscheidungen)
- Diese Idee wollen wir in unseren Algorithmus integieren
- Dafür brauchen wir problemabhängige Regeln, die eine Teilbelegung erweitern
- Beispiele zu Knapsack in der Hausaufgabe!

Propagation: Fixpunktiteration

Oft hat man mehrere Propagationsregeln:

- Teilweise werden Variablen komplett festgelegt
- Teilweise werden auch nur Wertebereiche eingeschränkt
- Teilweise werden auch zusätzliche Nebenbedingungen entdeckt
  (z.B. Gleichheit zwischen Variablen, …)
- Solange wenigstens eine Regel weitere Einschränkungen gefunden hat,
  betrachte alle Regeln ein weiteres Mal
- Stoppe erst, wenn keine weiteren Änderungen gefunden werden (Fixpunktiteration)
- Durch Propagation kann oft auch Unzulässigkeit festgestellt werden (Abbruch)
- Wenn es billige und teurere Regeln gibt:
  Nutze zuerst billige, und teurere Regeln erst sobald billige nicht mehr weiterhelfen

Propagation: Beispiele bei Sudoku

Beispiele für Regeln (wir haben zu jeder Zelle eine Menge zulässiger Werte):

- Wenn eine Zelle nur einen zulässigen Wert hat, kann man diesen aus anderen Zellen derselben Zeile, Spalte oder desselben Blocks entfernen.

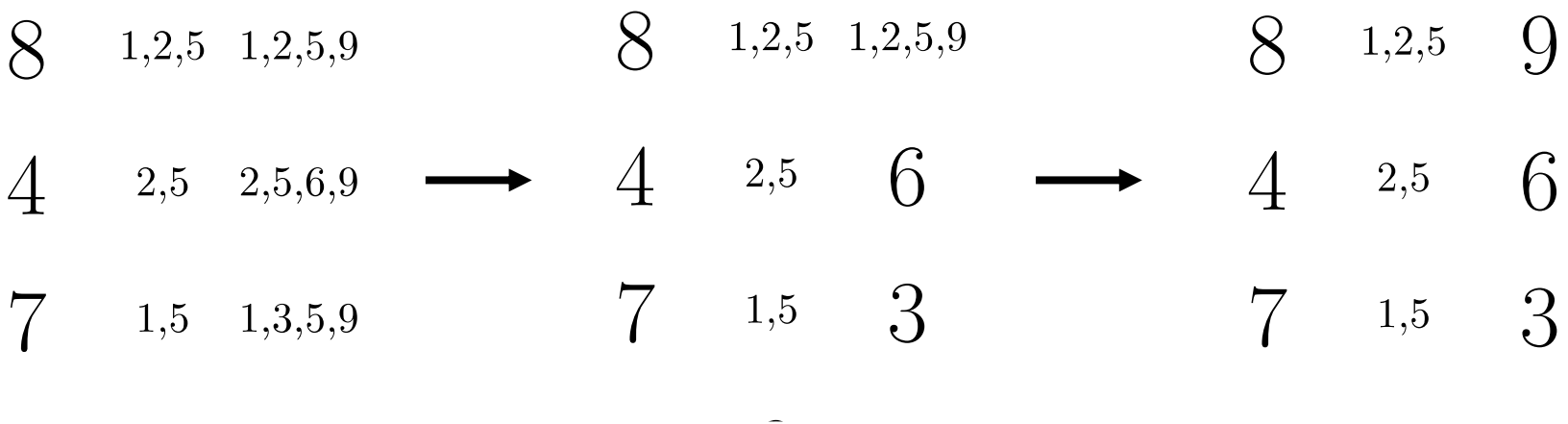
5	3	1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9	7	1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9

5	3	1, 2, 4, 6, 8, 9	1, 2, 4, 6, 8, 9	7	1, 2, 4, 6, 8, 9	1, 2, 4, 6, 8, 9	1, 2, 4, 6, 8, 9	1, 2, 4, 6, 8, 9

Propagation: Beispiele bei Sudoku

Beispiele für Regeln (wir haben zu jeder Zelle eine Menge zulässiger Werte):

- Wenn eine Zeile, eine Spalte oder ein Block nur eine Zelle mit möglichem Wert x hat, kann man alle anderen Werte aus dieser Zelle entfernen.

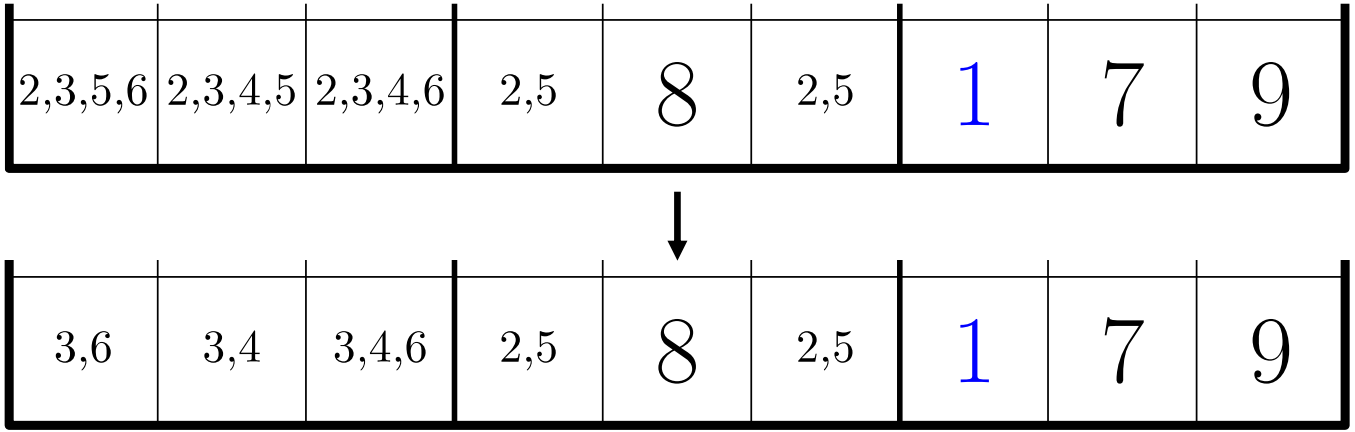


Propagation: Beispiele bei Sudoku

Paare:

- Wenn in einer Zeile/Spalte/Block 2 Werte nur in 2 Zellen auftreten, kann man alle anderen Werte aus diesen Zellen entfernen.
- Wenn in 2 Zellen nur 2 verschiedene Werte auftreten, kann man diese Werte aus allen anderen Zellen entfernen.

Tripel, Quadrupel, …: Verallgemeinerung von Paaren auf mehr Elemente



Erweiterter Algorithmus (Backtracking mit Propagation)

Algorithmus BTP
Voraussetzung: Variablen $x_0, \ldots, x_{n-1}$, Nebenbedingungen, Zulässigkeitstest Z, Propagation
Parameter: Array von Mengen erlaubter Werte $\mathcal{D} = [D(x_0), \ldots, D(x_{n-1})]$
Rückgabe: zulässige Lösung oder $\perp$

Wende Propagation an, um $\mathcal{D}$ einzuschränken
if $\exists x_i$ mit $D(x_i) = \emptyset$ **or not** $Z(\mathcal{D})$: **return** $\perp$
if $\forall x_i \ |D(x_i)| = 1$:
    if $\mathcal{D}$ kodiert Lösung: **return** $\mathcal{D}$
    else: return $\perp$
Wähle Variable x_i mit $|D(x_i)| > 1$
for each $v \in D(x_i)$:
    Erzeuge Kopie $\mathcal{D}' = \mathcal{D}$
    Setze $\mathcal{D}'[i] = \{v\}$
    $r = BTP(\mathcal{D}')$
    if $r \neq \perp$: **return** r
return $\perp$

Triff eine Entscheidung

Beispiel: Sudoku

Hier ein etwas schwereres Beispiel:

5		9						
	7				6			1
6				9				4
					9		5	
	4				3		8	
	5	8		7			2	
	6		2				4	
	8				7			3

Prop.

Basisreg.
Paare
Tripel

5	1,2,3	9	1,3,4,7,8	1,2,3,4,8	1,2,4,8	2,6,7,8	3,6,7	2,8
8	7	4	3,5	2,3,5	6	2,5	9	1
6	1,2,3	1,2,3	1,3,5,7,8	9	1,2,5,8	2,5,7,8	3,7	4
1,2,3,7	1,2,3	1,2,3,6,7	1,4,6,8	1,2,4,6,8	9	3,4	5	6,7
2,7,9	4	2,6,7	5,6	2,5,6	3	1	8	6,7,9
1,3,9	5	8	1,4,6	7	1,4	3,4	2	6,9
1,3,7	6	1,3,5,7	2	1,3,5,8	1,5,8	9	4	5,8
1,2,4	8	1,2,5	9	1,4,5,6	7	2,5,6	1,6	3
1,2,3,4,7	9	1,2,3,5,7	1,3,4,5,6,8	1,3,4,5,6,8	1,4,5,8	2,5,6,7,8	1,6,7	2,5,8

Backtracking & Propagation im Beispiel

5	1,2,3	9	1,3,4, 7,8	1,2,3, 4,8	1,2, 4,8	2,6, 7,8	3,6,7	2,8
8	7	4	3,5	2,3,5	6	2,5	9	1
6	1,2,3	1,2,3	1,3,5, 7,8	9	1,2, 5,8	2,5, 7,8	3,7	4
1,2, 3,7	1,2,3	1,2,3, 6,7	1,4, 6,8	1,2,4, 6,8	9	3,4	5	6,7
2,7,9	4	2,6,7	5,6	2,5,6	3	1	8	6,7,9
1,3,9	5	8	1,4,6	7	1,4	3,4	2	6,9
1,3,7	6	1,3, 5,7	2	1,3, 5,8	1,5,8	9	4	5,8
1,2,4	8	1,2,5	9	1,4, 5,6	7	2,5,6	1,6	3
1,2,3, 4,7	9	1,2,3, 5,7	1,3,4, 5,6,8	1,3,4, 5,6,8	1,4, 5,8	2,5,6, 7,8	1,6,7	2,5,8

Propagiere 2

Leere Mengen entstehen!

Propagiere 8

5	3	9	4	1	2	7	6	8
8	7	4	3	5	6	2	9	1
6	1	2	7	9	8	5	3	4
1	2	3	8	6	9	4	5	7
7	4	6	5	2	3	1	8	9
9	5	8	1	7	4	3	2	6
3	6	7	2	8	1	9	4	5
2	8	5	9	4	7	6	1	3
4	9	1	6	3	5	8	7	2

Probleme von Backtracking & Propagation

Der naive Algorithmus hat einige Schwachpunkte:

- Was, wenn eine relativ kleine Teilmenge der Variablen keine zulässige Belegung hat?
- Wir stoßen evtl. sehr oft (in vielen Teilbäumen) auf diesen Konflikt!
- Wir lernen nicht dazu!
- Wenn wir zuerst die richtigen Variablen betrachten würde, wäre es besser.
- Aber diese Variablenmenge ist schwer zu finden…

Wie könnten wir hinzulernen?

- Analysieren, wie es zu einem Konflikt kam...
- Daraus neue Nebenbedingungen lernen…
- Diese könnten global nützlich sein!
- Wie kann so etwas funktionieren?

Exkurs: SAT Solvers

Es gibt für einige schwere Probleme sehr gut optimierte Programme (Solver), u.A.:

- (Mixed) Integer Programming (Basis: LP, Branch & Bound + Erweiterungen)
- Constraint Programming (Basis: unterschiedliche Techniken, meist Backtracking)
- SAT (Boolesches Erfüllbarkeitsproblem)

SAT: Gegeben n Boolesche Variablen (wahr/falsch) $x_1, \dots, x_n$, und Formel φ in CNF

- Formel ist Konjunktion (UND-Verknüpfung) von *Klauseln*
- Jede Klausel ist Disjunktion (ODER-Verknüpfung) von *Literalen*
- Jedes Literal ist entweder x_i oder $\overline{x_i}$ (Variable oder negierte Variable)
- Also z.B. $\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$
- Gesucht ist eine *erfüllende Belegung* der x_i
- Eine Belegung der Variablen, die in jeder Klausel mindestens ein Literal wahr macht

Grundlage: Backtracking & Propagation

Angenommen, wir haben eine Teilbelegung der Variablen.

Jedes Literal (x_i bzw. $\overline{x_i}$) ist dadurch entweder *wahr*, *falsch* oder *offen*.

Was für Propagationsregeln und Zulässigkeitstests gibt es?

- **Empty Clause:** Alle Literale einer Klausel falsch ⇒ Teilbelegung unzulässig.
- **Unit Propagation (UP):** Wenn es eine Klausel gibt, in der alle Literale bis auf eines *falsch* sind, muss das verbleibende Literal *wahr* sein.
- **Beispiel:** Angenommen, x_1 und $\overline{x_2}$ sind wahr und wir hätten eine Klausel $x_2 \vee x_3 \vee \overline{x_1}$. Dann muss auch x_3 wahr sein!
- Gibt noch weitere mögliche Regeln, aber in der Praxis wird hauptsächlich UP genutzt.

Zentrale Datenstruktur: Trail

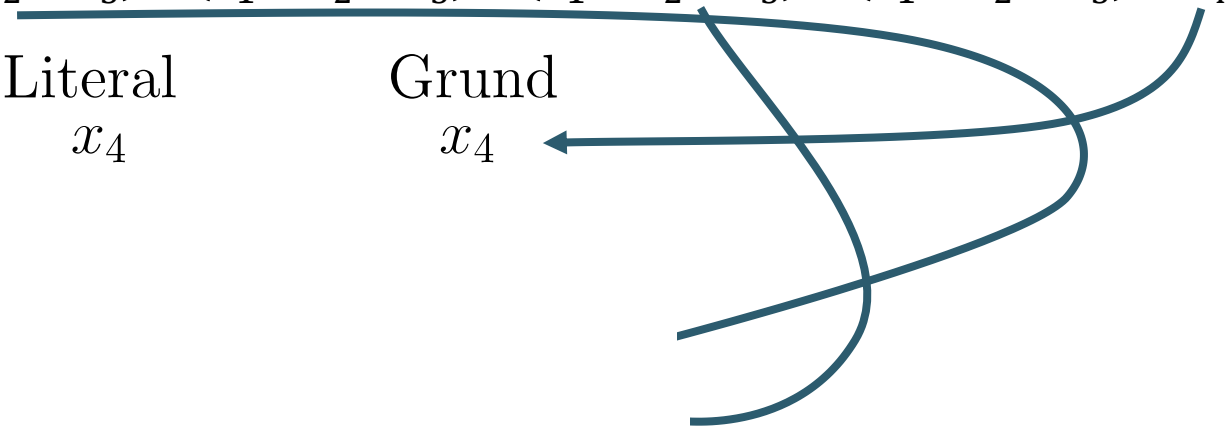
Praktische Implementierungen von Backtracking vermeiden oft Kopien der Teilbelegung

- Haben eine Datenstruktur, die Teilbelegungen verwaltet
  - Unterstützt das Hinzufügen einer Entscheidung inklusive Propagation
  - Unterstützt Zurücknehmen von Entscheidungen (inklusive resultierender Propagation)
- Name der Datenstruktur bei CDCL: *Trail*
- Zentrale Komponente: Stack wahrer Literale zusammen mit Gründen
- Unterstützt auch effizient Abfragen wie ‘ist x_i wahr/falsch/offen‘ etc.

Beispiel-Formel mit möglichem Trail

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge x_4$$

Level	Literal	Grund
0	x_4	x_4

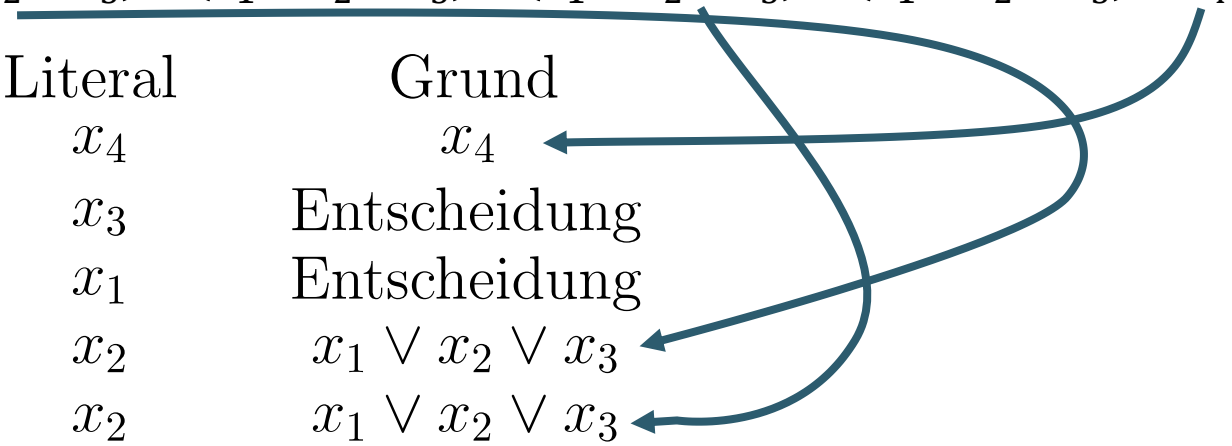


Konflikt!

Beispiel-Formel mit möglichem Trail

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge x_4$$

Level	Literal	Grund
0	x_4	x_4
1	x_3	Entscheidung
2	x_1	Entscheidung
2	x_2	$x_1 \vee x_2 \vee x_3$
2	x_2	$x_1 \vee x_2 \vee x_3$



Konflikt!

Konflikte behandeln

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge x_4$$

Ein Konflikt bedeutet, dass es mit unseren Entscheidungen keine erfüllende Belegung gibt.

- Im Beispiel gibt es also keine erfüllende Belegung mit $\overline{x_1} \wedge \overline{x_3}$.
- Das heißt, eines der Literale $\overline{x_1}, \overline{x_3}$ muss falsch sein, bzw. als Klausel $x_1 \vee x_3$.
- Hinzufügen dieser Klausel ändert nichts an den erfüllenden Belegungen.
- Hätten wir sie von Beginn an gehabt, sähe unser Trail wie folgt aus:

Level	Literal	Grund
0	x_4	x_4
1	x_3	Entscheidung
1	x_1	$x_1 \vee x_3$
1	x_2	$x_1 \vee x_2 \vee x_3$

Konflikte behandeln

Allgemein könnten wir Konflikte wie folgt behandeln:

- Wenn ein Konflikt auftritt, füge eine Klausel ein mit der Negation der Entscheidungen
- Gehe ein Entscheidungslevel zurück
- Auf diesem Level sorgt die neue Klausel dafür,
  dass Unit Propagation ein neues Literal produziert
- Solche Klauseln heißen *Konfliktklauseln*
- Dieses Literal ist die Negation der zurückgenommenen Entscheidung

Damit läuft der Algorithmus aber komplett analog zu Backtracking.

- Wir lernen nicht wirklich etwas neues dazu.
- Auch Backtracking trifft die genau gleiche Menge von Entscheidungen nur einmal!
- Gibt es eine Möglichkeit, ‘bessere’ Konfliktklauseln zu erzeugen?

Resolution

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge x_4$$

Sieht man der Formel irgendwie anders an, dass auch $x_1 \vee x_3$ gelten muss?

- Der Konflikt entstand durch die zwei Klauseln $(x_1 \vee x_2 \vee x_3)$ und $(x_1 \vee \overline{x_2} \vee x_3)$
- Zwei Fälle: $x_2 \rightarrow x_1 \vee x_3$, $\overline{x_2} \rightarrow x_1 \vee x_3$; insgesamt also $x_1 \vee x_3$!
- Allgemein: Resolution
- Zwei Klauseln $(x_i \vee \ell_1 \vee \cdots \vee \ell_p)$, $(\overline{x_i} \vee \mu_1 \vee \cdots \vee \mu_k)$ implizieren immer, dass auch $(x_i \vee \ell_1 \vee \cdots \vee \ell_p) \diamond (\overline{x_i} \vee \mu_1 \vee \cdots \vee \mu_k) = (\ell_1 \vee \cdots \vee \ell_p \vee \mu_1 \vee \cdots \vee \mu_k)$ gelten muss
- Dies nennt man (binäre) Resolution.
- Weitere Beispiele:

$$(x_1 \vee x_2 \vee x_3) \diamond (\overline{x_1} \vee x_2 \vee x_3) = (x_2 \vee x_3)$$
$$(x_1 \vee x_2 \vee x_3) \diamond (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) = (x_2 \vee x_3 \vee \overline{x_2} \vee \overline{x_3})$$

Wie kann uns das helfen?

Komplexeres Beispiel

Frage: Gibt es eine Folge k von 0en und 1en, sodass keine drei 0en und keine drei 1en im gleichen Abstand sind?

Also z.B. nicht 000 (zweimal Abstand 1), nicht 10101 (zweimal Abstand 2), nicht 0010110 (zweimal Abstand 3), …

Wir betrachten das Beispiel $k = 9$.

Kann man als SAT-Formel kodieren:

- Variablen $x_1, \ldots, x_k$, x_i sagt ob an Position i eine 0 (falsch) oder 1 (wahr) steht
- Welche Klauseln haben wir (z.B. für $k = 9$)?
  - $123, \overline{123}, 234, \overline{234}, \ldots$ (nicht zwei mal Abstand 1) (hier i statt x_i geschrieben)
  - $135, \overline{135}, 246, \overline{246}, \ldots$ (nicht zwei mal Abstand 2)
  - $147, \overline{147}, 258, \overline{258}, 369, \overline{369}$ (nicht zwei mal Abstand 3)
  - $159, \overline{159}$ (nicht zwei mal Abstand 4)

Beispiel

Level	Literal	Grund Entscheidung
1	6	

Konflikt:

- Klauseln $\overline{246}, 258$ Gründe für Konflikt
- Verbinde durch Resolution: $\overline{46}58$
- Hilft uns diese Klausel?
- Ändert sie irgendwas am Ablauf, wenn wir sie direkt gehabt hätten?
- Nicht unmittelbar! Auf jedem Level unter 3 gibt es mindestens 2 offene Literale!
- $\overline{46}58 \diamond \overline{468} = \overline{46}5$
- $\overline{46}5 \diamond \overline{456} = \overline{46}$
- Nur ein offenes Literal auf Level 1!
- Lerne $\overline{46}$; wie geht es jetzt weiter?

Beispiel

Level	Literal	Grund
1	6	Entscheidung
2	9	Entscheidung
2	3	369
3	4	Entscheidung
3	5	456
3	2	246
3	8	468
3	7	357
3	2	258

Konflikt:

- Klauseln $\overline{246}, 258$ Gründe für Konflikt
- Verbinde durch Resolution: $\overline{46}58$
- Hilft uns diese Klausel?
- Ändert sie irgendwas am Ablauf, wenn wir sie direkt gehabt hätten?
- Nicht unmittelbar! Auf jedem Level unter 3 gibt es mindestens 2 offene Literale!
- $\overline{46}58 \diamond \overline{468} = \overline{46}5$
- $\overline{46}5 \diamond \overline{456} = \overline{46}$
- Nur ein offenes Literal auf Level 1!
- Lerne $\overline{46}$; wie geht es jetzt weiter?

Verfahren zur Konfliktresolution

Konstruiere aus den zwei Gründen für die widersprüchlich belegte Variable eine Klausel C.

Solange C mehr als ein Literal aus dem aktuellen Level enthält:

- Gehe im Trail zum vorherigen Literal ℓ
- Falls $\overline{\ell} \in C$:
  - Sei R der Grund von ℓ
  - Wende Resolution an, setze $C = C \diamond R$
- Falls die triviale Klausel T (Negation der Entscheidungen) in C enthalten ist, setze $C = T$
- Lerne C als neue Klausel
- Gehe zurück zu dem Level, auf dem C durch Propagation ein neues Literal hinzugefügt hätte

Fortsetzung: Beispiel

Als hätten wir $\overline{46}$ zu Beginn gehabt!
Rücksprung zu Level 1!

Level	Literal	Grund
1	6	Entscheidung
1	4	46
2	5	Entscheidung
2	7	567
2	1	147
2	9	159
2	3	135
2	8	789
2	2	234
2	2	258

- $234 \diamond \overline{258} = 34\overline{58}$
- $34\overline{58} \diamond 789 = 34\bar{5}79$
- $34\bar{5}79 \diamond \overline{135} = \bar{1}4\bar{5}79$
- $\bar{1}4\bar{5}79 \diamond \overline{159} = \bar{1}4\bar{5}7$
- $\bar{1}4\bar{5}7 \diamond 147 = 4\bar{5}7$
- $4\bar{5}7 \diamond \overline{567} = 4\overline{56}$
- Mit dem unmodifizierten Verfahren lernen wir $4\overline{56}$.
- Können auch $\overline{56}$ lernen (wäre hier besser und ist die ‘triviale’ Backtracking-Klausel).
- Fortsetzung: Rücksprung zu Level 1. Nutze $\overline{56}$, um hier auch $\bar{5}$ abzuleiten. Wende dann Propagation an.

Fortsetzung: Beispiel

Level	Literal	Grund
1	6	Entscheidung
1	$\overline{4}$	$\overline{46}$
1	$\overline{5}$	$\overline{56}$
1	3	345
1	$\overline{9}$	$\overline{369}$
1	1	159
1	7	579
1	$\overline{2}$	$\overline{123}$
1	$\overline{8}$	$\overline{678}$
1	8	258

- $258 \diamond \overline{678} = 25\overline{67}$
- $25\overline{67} \diamond \overline{123} = \overline{13}5\overline{67}$
- $\overline{13}5\overline{67} \diamond 579 = \overline{13}5\bar{6}9$
- $\overline{13}5\bar{6}9 \diamond 159 = \bar{3}5\bar{6}9$
- $\bar{3}5\bar{6}9 \diamond \overline{369} = \bar{3}5\bar{6}$
- $\bar{3}5\bar{6} \diamond 345 = 45\bar{6}$
- $45\bar{6} \diamond \overline{56} = 4\bar{6}$
- $4\bar{6} \diamond \overline{46} = \bar{6}$
- Rücksprung zu Level 0!

Fortsetzung: Beispiel

Level	Literal	Grund
0	6	6
1	5	Entscheidung
1	4	456
1	7	567
1	1	147
1	3	135
1	9	159
1	2	234
1	8	789
1	8	258

- $258 \diamond \overline{789} = 25\overline{79}$
- $25\overline{79} \diamond \overline{234} = \overline{34}5\overline{79}$
- $\overline{34}5\overline{79} \diamond 159 = 1\overline{34}5\bar{7}$
- $1\overline{34}5\bar{7} \diamond 135 = 1\bar{4}5\bar{7}$
- $1\bar{4}5\bar{7} \diamond \overline{147} = \bar{4}5\bar{7}$
- $\bar{4}5\bar{7} \diamond 567 = \bar{4}56$
- $\bar{4}56 \diamond 456 = 56$
- Die triviale Klausel 5 ist wieder Teilmenge der unmodifiziert gelernten Klausel 56.
- Wir lernen 5 (allerdings spielt das hier keine große Rolle).

Fortsetzung: Beispiel

Level	Literal	Grund
0	6	6
0	5	5
1	4	Entscheidung
1	3	345
1	9	369
1	1	159
1	7	579
1	2	123
1	8	678
1	8	258

- $\overline{258} \diamond 678 = \overline{2567}$
- $\overline{2567} \diamond 123 = 13\bar{5}\bar{6}\bar{7}$
- $13\bar{5}\bar{6}\bar{7} \diamond \overline{579} = 13\bar{5}\bar{6}\bar{9}$
- $13\bar{5}\bar{6}\bar{9} \diamond \overline{159} = 3\bar{5}\bar{6}\bar{9}$
- $3\bar{5}\bar{6}\bar{9} \diamond 369 = 3\bar{5}\bar{6}$
- $3\bar{5}\bar{6} \diamond \overline{345} = \overline{456}$
- Wir können wieder $\bar{4}$ statt $\overline{456}$ lernen.
- Rücksprung wieder auf Level 0!

Fortsetzung: Beispiel

Level	Literal	Grund
0	6	6
0	5	5
0	4	4
0	2	246
0	8	468
0	8	258

- Konflikt auf Level 0!
- Damit hat die Formel keine erfüllende Belegung (UNSAT)!
- Als Zertifikat/Beweis leere Klausel konstruieren:
  - $\overline{258} \diamond 468 = \overline{25}46$
  - $\overline{25}46 \diamond 246 = \bar{5}46$
  - $\bar{5}46 \diamond \bar{4} = \bar{5}6$
  - $\bar{5}6 \diamond 5 = 6$
  - $6 \diamond \bar{6} = \square$
- Es gibt also keine solche Sequenz der Länge $k = 9$.

Algorithmus CDCL

Conflict Driven Clause Learning (CDCL):

1. Propagiere, triff Entscheidungen, propagiere…
2. Falls dabei ohne Konflikte alle Variablen belegt werden → SAT, Belegung gefunden.
3. Falls Konflikt auftritt: Analysiere Konflikt, lerne Konfliktklausel
4. Springe zurück auf das Level, wo neue Klausel durch Propagation neue Literale ergibt
5. Propagiere und wiederhole, solange dabei erneut Konflikt auftritt; sonst zurück zu 1.
6. Falls Konflikt auf Level 0 auftritt → UNSAT.

Die Idee ist der Kern moderner SAT-Solver.

Sie ist dort sehr erfolgreich und löst regelmäßig auch sehr große SAT-Instanzen.

Jede Menge offene Punkte und Parameter (es gibt viele verschiedene SAT-Solver auf CDCL-Basis) und jede Menge mögliche Erweiterungen.