DPLL-АЛГОРИТМ

Кузнецова Арина 6373

- Алгоритм Дэвиса–Патнема–Логемана–Лавленда (DPLL) был разработан в 1962 году для определения выполнимости булевых формул, записанных в конъюнктивной нормальной форме, т.е. для решения задачи SAT. Алгоритм оказался настолько эффективным, что спустя уже более 50 лет представляет собой основу для большинства эффективных решателей SAT.

Суть алгоритма

Давайте разберем подробнее, что же делает алгоритм DPLL. Он берет булеву формулу и пытается разделить все переменные, входящие в нее, на два множества A и B, где A — множество всех переменных со значением true, а B — множество всех переменных со значением false.

На каждом шаге некоторым образом выбирается переменная, которой еще не присвоено значение (назовем такие переменные *свободными*) и присваивается значение true (эта переменная заносится в множество A). После этого решается полученная упрощенная задача. Если она выполнима, то и исходная формула выполнима. Иначе — выбранной переменной присваивается значение false (она заносится в B) и задача решается для новой упрощенной формулы. Если она выполнима, то и исходная формула выполнима. Иначе — увы, исходная формула невыполнима.

После каждого присваивания формула дополнительно упрощается при помощи следующих двух правил:

- Распространение переменной (unit propagation). Если в какой-либо дизъюнкте осталась ровно одна переменная, то ей необходимо присвоить такое значение, чтобы дизъюнкта в итоге стала истинной (переместить в A или B в зависимости от того, есть отрицание или нет).
- Исключение «чистых» переменных (pure literal elimination). Если какая-либо переменная входит в формулу только с отрицаниями, либо всегда без отрицаний — она называется *чистой*. Этой переменной можно присвоить такое значение, что все ее вхождения будут иметь значение true, что уменьшит число свободных переменных.

Данные два правила следует применять до тех пор, пока они применяются: обычно после первого присваивания следует целый каскад упрощений, что хорошо уменьшает число свободных переменных.

Если после упрощения мы получили пустую дизъюнкту (все ее простые конъюнкты ложны) — текущая формула не выполнима и следует откатиться. Если же свободных переменных не осталось — то формула выполнима и работу алгоритма можно закончить. Также закончить работу алгоритма можно в том случае, если дизъюнкт не осталось — неиспользованные свободные переменные можно назначить произвольным образом.

- Рассмотрим следующую формулу и на ее примере посмотрим как работает DPLL-алгоритм:

$$(x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$$
$$A = \varnothing, B = \varnothing, C = \{x_1, x_2, x_3, x_4\}$$

- К этой формуле не применимы правила упрощения, поэтому придется ветвить наше дерево. В качестве элемента для ветвления возьмем x_1 и, для начала, присвоим ему значение true. Получим следующую цепочку упрощений:

$$(\neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$$
$$A = \{x_1\}, B = \varnothing, C = \{x_2, x_3, x_4\}$$

$$\Big\downarrow\, A \leftarrow x_3$$

$$(\neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (x_2 \vee x_4)$$
$$A = \{x_1, x_3\}, B = \varnothing, C = \{x_2, x_4\}$$

$$\Big\downarrow\, B \leftarrow x_4$$

$$(\neg x_2) \wedge (x_2)$$
$$A = \{x_1, x_3\}, B = \{x_4\}, C = \{x_2\}$$

$$\Big\downarrow\, B \leftarrow x_2$$

$$()$$
$$A = \{x_1, x_3\}, B = \{x_2, x_4\}, C = \varnothing$$

Fail!

- Двойные стрелочки показывают, что мы используем первое правило, а именно — находим одинокую переменную и присваиваем ей нужное значение.

  К сожалению, данная ветвь ведет к невыполнимой формуле, т.е. в этой ветви мы зря старались. Откатываемся и пробуем присвоить переменной x_1 значение false. Это приведет к следующей цепочке упрощений:

$$\boxed{\begin{array}{c}(x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \\ A = \varnothing, B = \{x_1\}, C = \{x_2, x_3, x_4\}\end{array}}$$

$$\Downarrow \; B \leftarrow x_3$$

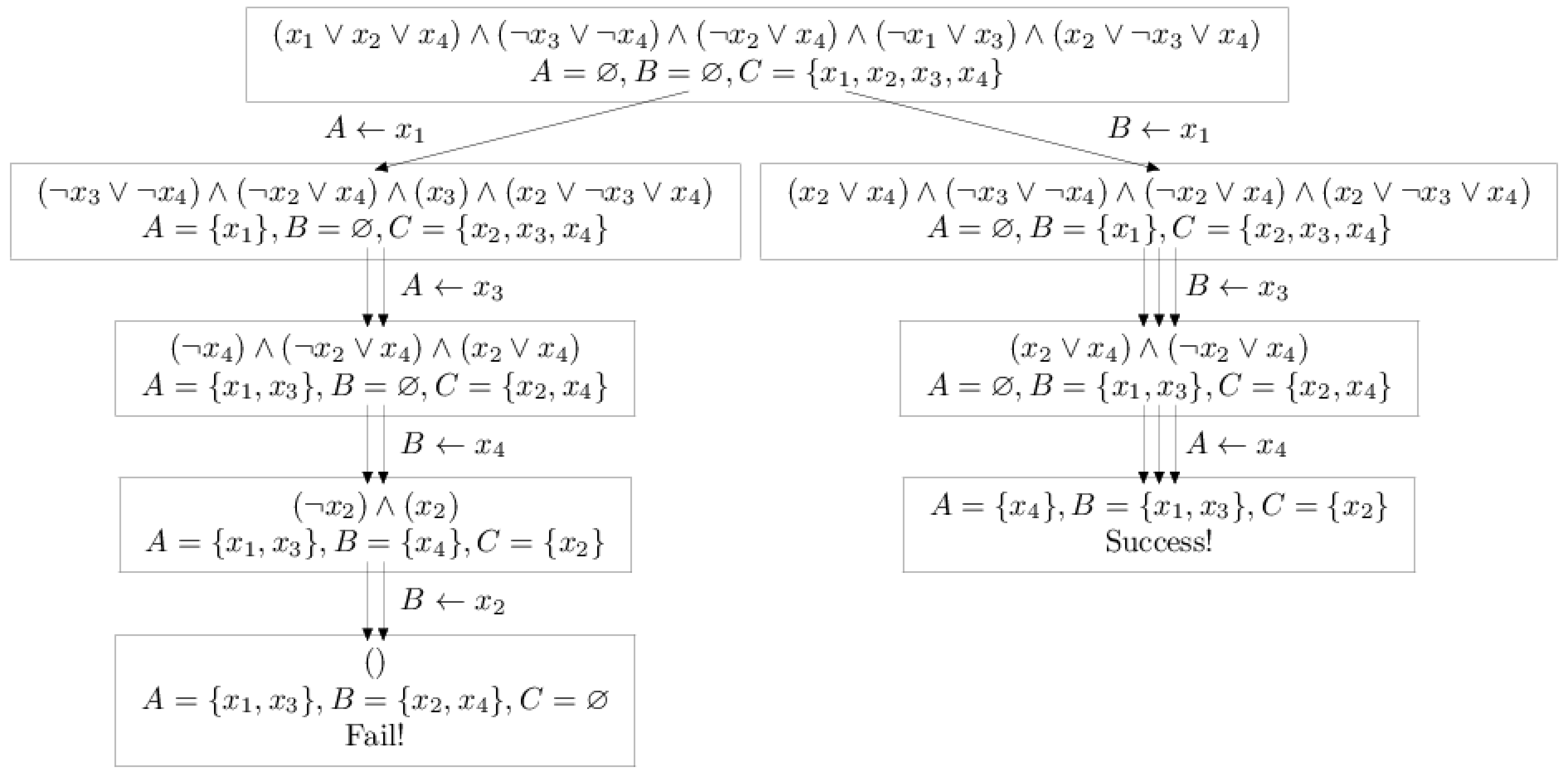
$$\boxed{\begin{array}{c}(x_2 \vee x_4) \wedge (\neg x_2 \vee x_4) \\ A = \varnothing, B = \{x_1, x_3\}, C = \{x_2, x_4\}\end{array}}$$

$$\Downarrow \; A \leftarrow x_4$$

$$\boxed{\begin{array}{c}A = \{x_4\}, B = \{x_1, x_3\}, C = \{x_2\} \\ \text{Success!}\end{array}}$$

- Тройные стрелки показывают, что мы применяем второе правило упрощения. В этой ветви нас ждет успех — найдено целых 2 решения.

- Дерево обхода целиком:



Стоит также отметить, что с помощью данного алгоритма нельзя найти все решения — этому мешает эвристика исключения «чистых» переменных. Вполне может оказаться пропущено решение, в котором значение той переменной, которую мы, следуя эвристике, установили в true, равно false. Для поиска всех решений нужно исключить второе правило из алгоритма.

В чём же выигрыш этого алгоритма перед простым перебором?

Следуя из вышеописанного, в ходе выполнения алгоритма многие выражения могут отсекаться (не рассматриваться), так как если одна из переменных в дизъюнктивном выражении равна 1, то и всё выражение равно 1 и можно в нём остальные переменные не рассматривать. Это достаточно сильно ускоряет работу алгоритма, хотя в худшем случае он всё равно работает за экспоненциальное время (2^n, где n - количество переменных).