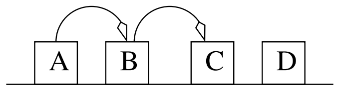
Parallel plans

- Plans are not sequences $o_1, \ldots, o_n$ of operators, but sequences $S_1, \ldots, S_n$ of *sets of operators*.
- All operators at a given step are applied simultaneously.
- Requirement: result of simultaneous application must be the same as *application in any order* (interleaving semantics)

Parallel plans: example

Simultaneous actions possible (actions do not *interfere*):

Not possible (B not movable when A is on top of it):



Parallel plans

Let S be a set of operators and s a state.

Define *app*$_S(s)$ as the result of simultaneously applying all operators $o \in S$ in state s:

1. the preconditions of all operators in S must be true in s, and
2. the state *app*$_S(s)$ is obtained from s by making the literals in $\bigcup_{\langle p,e\rangle \in S} ([e]_s)$ true.

Parallel plans

For a set of operators O and an initial state I, *a parallel plan* is a sequence $T = S_1, \ldots, S_l$ of sets of operators such that there is a sequence of states $s_0, \ldots, s_l$ (the execution of T) such that

1. $s_0 = I$,
2. $\bigcup_{\langle p,e\rangle \in S_i} ([e]_{s_{i-1}})$ is consistent for every $i \in \{1, \ldots, l\}$,
3. $s_i = \mathit{app}_{S_i}(s_{i-1})$ for $i \in \{1, \ldots, l\}$,
4. for all $i \in \{1, \ldots, l\}$ and $\langle p, e\rangle = o \in S_i$ and $S \subseteq S_i \backslash \{o\}$,
   (a) $\mathit{app}_S(s_{i-1}) \models p$ and
   (b) $[e]_{s_{i-1}} = [e]_{\mathit{app}_S(s_{i-1})}$.

Parallel plans

LEMMA A Let $T = S_1, \ldots, S_k, \ldots, S_l$ be a parallel plan. Let $T' = S_1, \ldots, S_k^0, S_k^1, \ldots, S_l$ be the parallel plan obtained from T by splitting the step S_k into two steps S_k^0 and S_k^1 such that $S_k = S_k^0 \cup S_k^1$ and $S_k^0 \cap S_k^1 = \emptyset$.

If $s_0, \ldots, s_k, \ldots, s_l$ is the execution of T then $s_0, \ldots, s'_k, s_k, \ldots, s_l$ for some s'_k is the execution of T'.

Parallel plans

THEOREM Let $T = S_1, \ldots, S_k, \ldots, S_l$ be a parallel plan. Then any $\sigma = o_1^1; \ldots; o_{n_1}^1; o_2^2; \ldots; o_{n_2}^2; \ldots; o_1^l; \ldots; o_{n_l}^l$ such that for every $i \in \{1, \ldots, l\}$ the sequence $o_1^i; \ldots; o_{n_i}^i$ is a total ordering of S_i, is a plan, and its execution leads to the same terminal state as that of T.

PROOF: First, all empty steps can be removed from the parallel plan. By Lemma A non-singleton steps can be split repeatedly to two smaller non-empty steps until every step is singleton and the singleton steps are in the desired order.

Planning as satisfiability: parallel encoding

To obtain valid parallel plans, include in $\mathcal{R}_2(P, P')$ the formula

$$\neg(o_i \wedge o_j)$$

for every $o_i, o_j \in O$ such that $i \neq j$ and there is a state variable $p \in P$ such that

1. p occurs in an effect in o_i, and
2. p occurs in a formula in o_j (in the precondition or in the antecedent of a conditional effect in o_j)

Planning as satisfiability: parallel encoding

Reading the plan from a satisfying assignment v: for all $t \in \{1, \ldots, l\}$,

$$S_t = \{o \in O | v(o^t) = 1\}.$$

THEOREM $S_1, \ldots, S_l$ satisfies the definition of parallel plans.

PROOF IDEA: For every $S \subseteq S_i$, applying S does not change the values of the precondition or antecedents of conditionals of any operator in $S_i \backslash S$, because the state variables in the effects in S are disjoint from those in the formulae.

Conjunctive normal form

Many satisfiability algorithms require formulas in the conjunctive normal form: transformation by repeated applications of the following equivalences.

$$
\begin{aligned}
\neg(\phi \vee \psi) &\equiv \neg\phi \wedge \neg\psi \\
\neg(\phi \wedge \psi) &\equiv \neg\phi \vee \neg\psi \\
\neg\neg\phi &\equiv \phi \\
\phi \vee (\psi_1 \wedge \psi_2) &\equiv (\phi \vee \psi_1) \wedge (\phi \vee \psi_2)
\end{aligned}
$$

The formula is conjunction of *clauses* (disjunctions of literals).

EXAMPLE: $(A \vee \neg B \vee C) \wedge (\neg C \vee \neg B) \wedge A$

The unit resolution rule

From $l_1 \vee l_2 \vee \cdots \vee l_n$ (here $n \geq 1$) and $\overline{l_1}$ infer $l_2 \vee \cdots \vee l_n$.

EXAMPLE: From $A \vee B \vee C$ and $\neg A$ infer $B \vee C$.

SPECIAL CASE: from A and $\neg A$ we get the empty clause $\perp$ (*"disjunction consisting of zero disjuncts"*).

Satisfiability test by the Davis-Putnam procedure

1. Let C be a set of clauses.
2. For all clauses $l_1 \vee l_2 \vee \cdots \vee l_n \in C$ and $\overline{l_1} \in C$, remove $l_1 \vee l_2 \vee \cdots \vee l_n$ from C and add $l_2 \vee \cdots \vee l_n$ to C.
3. For all clauses $l_1 \vee l_2 \vee \cdots \vee l_n \in C$ and $l_1 \in C$, remove $l_1 \vee l_2 \vee \cdots \vee l_n$ from C. (UNIT SUBSUMPTION)
4. If $\perp \in C$, return FALSE.
5. If C contains only unit clauses, return TRUE.
6. Pick some $p \in P$ such that $\{p, \neg p\} \cap C = \emptyset$
7. Recursive call: if $C \cup \{p\}$ is satisfiable, return TRUE.
8. Recursive call: if $C \cup \{\neg p\}$ is satisfiable, return TRUE.
9. Return FALSE.

Planning as satisfiability: example

clear(C), on(C,B), on(B,A), ontable(A), clear(E), on(E,D), ontable(D) are initially true (there are two stacks, CBA and ED.)

The goal is on(A,B)$\wedge$on(B,C)$\wedge$on(C,D)$\wedge$on(D,E)

The Davis-Putnam procedure solves the problem quickly:

- Formulae for lengths 1 to 4 shown unsatisfiable by unit resolution.
- Formula for plan length 5 is satisfiable: 3 nodes in the search tree.

Planning as satisfiability: example

```
v0.9 13/08/1997 19:32:47
30 propositions 100 operators
Length 1
Length 2
Length 3
Length 4
Length 5
branch on -clear(b)[1] depth 0
branch on clear(a)[3] depth 1
Found a plan.
  0 totable(e,d)
  1 totable(c,b) fromtable(d,e)
  2 totable(b,a) fromtable(c,d)
  3 fromtable(b,c)
  4 fromtable(a,b)
Branches 2 last 2 failed 0; time 0.0
```



```
            012345   012345   012345
   clear(a) FF       FFF TT   FFFTTT
   clear(b) F    F   FF TTF   FFTTTF
   clear(c) TT  FF   TTTTFF   TTTTFF
   clear(d) FTTFFF   FTTFFF   FTTFFF
   clear(e) TTFFFF   TTFFFF   TTFFFF
    on(a,b) FFF  T   FFFFFT   FFFFFT
    on(a,c) FFFFFF   FFFFFF   FFFFFF
    on(a,d) FFFFFF   FFFFFF   FFFFFF
    on(a,e) FFFFFF   FFFFFF   FFFFFF
    on(b,a) TT  FF   TTT FF   TTTFFF
    on(b,c) FF  TT   FFFFTT   FFFFTT
    on(b,d) FFFFFF   FFFFFF   FFFFFF
    on(b,e) FFFFFF   FFFFFF   FFFFFF
    on(c,a) FFFFFF   FFFFFF   FFFFFF
    on(c,b) T  FFF   TT FFF   TTFFFF
    on(c,d) FFFTTT   FFFTTT   FFFTTT
```



```
    on(c,e) FFFFFF   FFFFFF   FFFFFF
    on(d,a) FFFFFF   FFFFFF   FFFFFF
    on(d,b) FFFFFF   FFFFFF   FFFFFF
    on(d,c) FFFFFF   FFFFFF   FFFFFF
    on(d,e) FFTTTT   FFTTTT   FFTTTT
    on(e,a) FFFFFF   FFFFFF   FFFFFF
    on(e,b) FFFFFF   FFFFFF   FFFFFF
    on(e,c) FFFFFF   FFFFFF   FFFFFF
    on(e,d) TFFFFF   TFFFFF   TFFFFF
ontable(a) TTT  F   TTTTTF   TTTTTF
ontable(b) FF  FF   FFF FF   FFFTFF
ontable(c) F  FFF   FF FFF   FFTFFF
ontable(d) TTFFFF   TTFFFF   TTFFFF
ontable(e) FTTTTT   FTTTTT   FTTTTT
```



```
               01234
fromtable(a,b) ....T
fromtable(b,c) ...T.
fromtable(c,d) ..T..
fromtable(d,e) .T...
  totable(b,a) ..T..
  totable(c,b) .T...
  totable(e,d) T....
```