## Distance estimation for heuristic search

- PROBLEM: How to compute good distance/cost estimates $h(s)$ for controlling heuristic search algorithms like A$*$, best-first search or local search algorithms?
- *If we knew the distances exactly, it would be very easy to choose one of the operators that takes us one step closer to a goal state.* (Computing exact distances is PSPACE-hard!)
- Compute *a lower bound* $\delta_s(G)$ on the number of operators needed to reach a goal state from $s$.



## Distance estimation: example, blocks world

We have three blocks initially with A on B and B on C:

$$
\begin{array}{rcl}
D_0 & = & \{\text{A-CLEAR}, \text{A-ON-B}, \text{B-ON-C}, \text{C-ON-TABLE}, \neg\text{A-ON-C}, \\
 & & \neg\text{B-ON-A}, \neg\text{C-ON-A}, \neg\text{C-ON-B}, \neg\text{A-ON-TABLE}, \\
 & & \neg\text{B-ON-TABLE}, \neg\text{B-CLEAR}, \neg\text{C-CLEAR}\} \\
D_1 & = & \{\text{A-CLEAR}, \text{B-ON-C}, \text{C-ON-TABLE}, \neg\text{A-ON-C}, \neg\text{B-ON-A}, \\
 & & \neg\text{C-ON-A}, \neg\text{C-ON-B}, \neg\text{B-ON-TABLE}, \neg\text{C-CLEAR}\} \\
D_2 & = & \{\text{C-ON-TABLE}, \neg\text{A-ON-C}, \neg\text{C-ON-A}, \neg\text{C-ON-B}\} \\
D_3 & = & \emptyset
\end{array}
$$



| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ |
|---|---|---|---|---|---|
| A-ON-B | $T$ | | | | |
| A-ON-C | $F$ | | | | |
| B-ON-A | $F$ | | | | |
| B-ON-C | $T$ | | | | |
| C-ON-A | $F$ | | | | |
| C-ON-B | $F$ | | | | |
| A-ON-TABLE | $F$ | | | | |
| B-ON-TABLE | $F$ | | | | |
| C-ON-TABLE | $T$ | | | | |
| A-CLEAR | $T$ | | | | |
| B-CLEAR | $F$ | | | | |
| C-CLEAR | $F$ | | | | |



| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ |
|---|---|---|---|---|---|
| A-ON-B | $T$ | $TF$ | | | |
| A-ON-C | $F$ | $F$ | | | |
| B-ON-A | $F$ | $F$ | | | |
| B-ON-C | $T$ | $T$ | | | |
| C-ON-A | $F$ | $F$ | | | |
| C-ON-B | $F$ | $F$ | | | |
| A-ON-TABLE | $F$ | $TF$ | | | |
| B-ON-TABLE | $F$ | $F$ | | | |
| C-ON-TABLE | $T$ | $T$ | | | |
| A-CLEAR | $T$ | $T$ | | | |
| B-CLEAR | $F$ | $TF$ | | | |
| C-CLEAR | $F$ | $F$ | | | |

| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ |
|---|---|---|---|---|---|
| A-ON-B | $T$ | $TF$ | $TF$ | | |
| A-ON-C | $F$ | $F$ | $F$ | | |
| B-ON-A | $F$ | $F$ | $TF$ | | |
| B-ON-C | $T$ | $T$ | $T$ | | |
| C-ON-A | $F$ | $F$ | $F$ | | |
| C-ON-B | $F$ | $F$ | $F$ | | |
| A-ON-TABLE | $F$ | $TF$ | $TF$ | | |
| B-ON-TABLE | $F$ | $F$ | $TF$ | | |
| C-ON-TABLE | $T$ | $T$ | $T$ | | |
| A-CLEAR | $T$ | $T$ | $TF$ | | |
| B-CLEAR | $F$ | $TF$ | $TF$ | | |
| C-CLEAR | $F$ | $F$ | $F$ | | |



| | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ |
|---|---|---|---|---|---|
| A-ON-B | $T$ | $TF$ | $TF$ | $TF$ | $TF$ |
| A-ON-C | $F$ | $F$ | $F$ | $TF$ | $TF$ |
| B-ON-A | $F$ | $F$ | $TF$ | $TF$ | $TF$ |
| B-ON-C | $T$ | $T$ | $T$ | $TF$ | $TF$ |
| C-ON-A | $F$ | $F$ | $F$ | $TF$ | $TF$ |
| C-ON-B | $F$ | $F$ | $F$ | $TF$ | $TF$ |
| A-ON-TABLE | $F$ | $TF$ | $TF$ | $TF$ | $TF$ |
| B-ON-TABLE | $F$ | $F$ | $TF$ | $TF$ | $TF$ |
| C-ON-TABLE | $T$ | $T$ | $T$ | $TF$ | $TF$ |
| A-CLEAR | $T$ | $T$ | $TF$ | $TF$ | $TF$ |
| B-CLEAR | $F$ | $TF$ | $TF$ | $TF$ | $TF$ |
| C-CLEAR | $F$ | $F$ | $F$ | $TF$ | $TF$ |



## Inaccuracy of the representation

Consider the initial state 0000 (with state variables D, E, F, G). $D_0 = \{\neg D, \neg E, \neg F, \neg G\}$ represents the states $\{$0000$\}$.

The operators are $O = \{\langle \neg D, E\rangle, \ \ \langle \neg E, D\rangle\}$.

Now $D_1 = \{\neg F, \neg G\}$, and it represents $\{$0000,0100,1000,1100$\}$.

However, the state 1100 is **not reachable** from 0000!



## The function makestrue$(l, O)$

$\phi \in$ makestrue$(l, O)$ if there is an operator in $O$ that is applicable and makes literal $l$ true whenever $\phi$ is true.

EXAMPLE: Let $o = \langle A \wedge B, R \wedge (Q \rhd C) \wedge (R \rhd C)\rangle$. Now

$$\text{makestrue}(C, \{o\}) = \{A \wedge B \wedge Q, \ \ A \wedge B \wedge R\}.$$

REMARK: For operators without conditional effects this is just the set of preconditions of those operators that make the literal true.

## The sets $D_0, D_1, \ldots$

Let $L = P \cup \{\neg p | p \in P\}$ be the set of literals on $P$.

Define the sets $D_i$ for $i \geq 0$ as follows.

$$
\begin{aligned}
D_0 &= \{l \in L | s \models l\} \\
D_i &= D_{i-1} \backslash \{l \in L | \phi \in \text{makestrue}(\bar{l}, O), \text{canbetrue}(\phi, D_{i-1})\}
\end{aligned}
$$

If $n = |P|$, then $D_n = D_{n+1}$, because at most $n$ times there can be a literal contained in $D_i$ but not in $D_{i+1}$.



## The procedure canbetrue$(\phi, D)$

canbetrue$(\phi, D)$ returns *true* whenever $D \cup \{\phi\}$ is satisfiable.

Equivalently: there is a state described by the literals in $D$ in which $\phi$ is true.

The procedure runs in *polynomial time* but satisfiability testing is NP-hard (known algorithms take exponential time).

The procedure *fails in one direction*: e.g. canbetrue$(A \wedge \neg A, \emptyset)$ returns true (*BUT does not invalidate distance estimation, which is not meant to be accurate anyway!!*)



## The procedure canbetrue$(\phi, D)$: definition

$$
\begin{aligned}
\text{canbetrue}(\bot, D) &= \text{false} \\
\text{canbetrue}(\top, D) &= \text{true} \\
\text{canbetrue}(p, D) &= \text{true iff } \neg p \notin D \quad \text{(for state variables } p \in P\text{)} \\
\text{canbetrue}(\neg p, D) &= \text{true iff } p \notin D \quad \text{(for state variables } p \in P\text{)} \\
\text{canbetrue}(\neg\neg\phi, D) &= \text{canbetrue}(\phi, D) \\
\text{canbetrue}(\phi \vee \psi, D) &= \text{canbetrue}(\phi, D) \text{ or canbetrue}(\psi, D) \\
\text{canbetrue}(\phi \wedge \psi, D) &= \text{canbetrue}(\phi, D) \text{ and canbetrue}(\psi, D) \\
\text{canbetrue}(\neg(\phi \vee \psi), D) &= \text{canbetrue}(\neg\phi, D) \text{ and canbetrue}(\neg\psi, D) \\
\text{canbetrue}(\neg(\phi \wedge \psi), D) &= \text{canbetrue}(\neg\phi, D) \text{ or canbetrue}(\neg\psi, D)
\end{aligned}
$$



## The procedure canbetrue$(\phi, D)$: correctness

LEMMA A

Let $\phi$ be a formula and $D$ a consistent set of literals (it contains at most one of $p$ and $\neg p$ for every $p \in P$.) If $D \cup \{\phi\}$ is satisfiable, then canbetrue$(\phi, D)$ returns true.

PROOF: by induction on the structure of $\phi$.

Base case 1, $\phi = \bot$: The set $D \cup \{\bot\}$ is not satisfiable, and hence the implication trivially holds.

Base case 2, $\phi = \top$: canbetrue$(\top, D)$ always returns true, and hence the implication trivially holds.

Base case 3, $\phi = p$ for some $p \in P$: If $D \cup \{p\}$ is satisfiable, then $\neg p \notin D$, and hence canbetrue$(p, D)$ returns true.

Base case 4, $\phi = \neg p$ for some $p \in P$: If $D \cup \{\neg p\}$ is satisfiable, then $p \notin D$, and hence canbetrue$(\neg p, D)$ returns true.

Inductive case 1, $\phi = \neg\neg\phi'$ for some $\phi'$: The formulae are logically equivalent, and by the induction hypothesis we directly establish the claim.

Inductive case 2, $\phi = \phi' \vee \psi'$: If $D \cup \{\phi' \vee \psi'\}$ is satisfiable, then either $D \cup \{\phi'\}$ or $D \cup \{\psi'\}$ is satisfiable and by the



induction hypothesis at least one of canbetrue$(\phi', D)$ and canbetrue$(\psi', D)$ returns true. Hence canbetrue$(\phi' \vee \psi', D)$ returns true.

Inductive case 3, $\phi = \phi' \wedge \psi'$: If $D \cup \{\phi' \wedge \psi'\}$ is satisfiable, then both $D \cup \{\phi'\}$ and $D \cup \{\psi'\}$ are satisfiable and by the induction hypothesis both canbetrue$(\phi', D)$ and canbetrue$(\psi', D)$ return true. Hence canbetrue$(\phi' \wedge \psi', D)$ returns true.

Inductive cases 4 and 5, $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

Q.E.D.



## Definition of distances for formulae

$$\delta_s(\phi) = \begin{cases} 0 & \text{if canbetrue}(\phi, D_0) \\ d & \text{if canbetrue}(\phi, D_d) \text{ and not canbetrue}(\phi, D_{d-1}) \text{ (for } d \end{cases}$$



## Definition of distances for formulae: correctness

LEMMA B

Let $s$ be a state and $D_0, D_1, \ldots$ the respective distance sets. If $s'$ is the state reached from $s$ by applying the operator sequence $o_1, \ldots, o_n$, then $s' \models D_n$.

PROOF: by induction on the length of the sequence.

Base case $n = 0$: The length of the operator sequence is zero, and hence $s' = s$. The set $D_0$ consists exactly of those literals that are true in $s$, and hence $s' \models D_0$.

Inductive case $n \geq 1$: Let $s''$ be the state reached from $s$ by applying $o_1, \ldots, o_{n-1}$. Now $s' = \text{app}_{o_n}(s'')$. By the induction hypothesis $s'' \models D_{n-1}$.

Let $l$ be any literal in $D_n$. We show that $s' \models l$. Because $l \in D_n$ and $D_n \subseteq D_{n-1}$, also $l \in D_{n-1}$, and hence by IH $s'' \models l$.

Let $\phi$ be any member of makestrue$(\bar{l}, \{o_n\})$. Because $l \in D_n$ it must be that canbetrue$(\phi, D_{n-1})$ returns false (Definition of $D_n$). Hence $D_{n-1} \cup \{\phi\}$ is by Lemma A not satisfiable, and $s'' \not\models \phi$. Hence applying $o_n$ in $s''$ does not make $l$ false, and finally $s' \models l$.

Q.E.D.



## Definition of distances for formulae: correctness

THEOREM

Let $s$ be a state, $\phi$ a formula, and $D_0, D_1, \ldots$ the respective distance sets. If $s'$ is the state reached from $s$ by applying the operators $o_1, \ldots, o_n$ and $s' \models \phi$ for any formula $\phi$, then canbetrue$(\phi, D_n)$ returns true.

PROOF

By Lemma B $s' \models D_n$. By assumption $s' \models \phi$. Hence $D_n \cup \{\phi\}$ is satisfiable. By Lemma A canbetrue$(\phi, D_n)$ returns true.

Q.E.D.



## Definition of distances for formulae: correctness

COROLLARY

Let $s$ be a state and $\phi$ a formula. Then for any sequence $o_1, \ldots, o_n$ of operators such that executing them in $s$ results in state $s'$ such that $s' \models \phi$, $n \geq \delta_s(\phi)$.

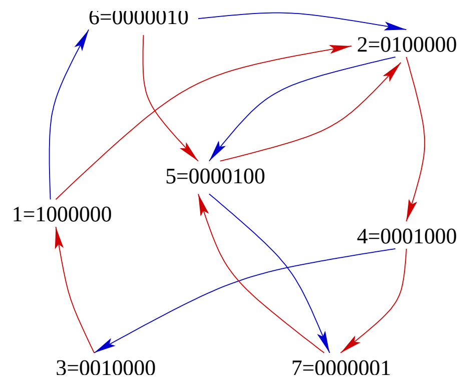
PROOF

By the previous result canbetrue$(\phi, D_n)$ returns *true*. Hence by definition $\delta_s(\phi) \leq n$.

Q.E.D.



## Distance estimation: example, distance 1 to 3



# Distance estimation: example, distance 1 to 3

Let the state variables be A, B, C, D, E, F, G.

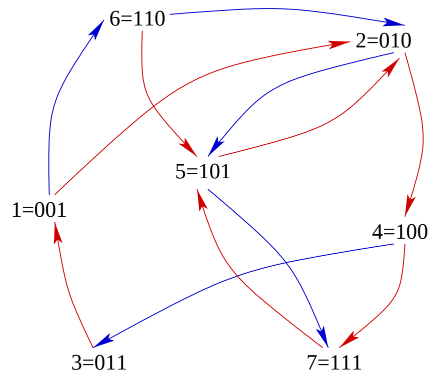
$$
\begin{aligned}
D_0 &= \{A, \neg B, \neg C, \neg D, \neg E, \neg F, \neg G\} \\
D_1 &= \{\neg C, \neg D, \neg E, \neg G\} \\
D_2 &= \{\neg C, \neg G\} \\
D_3 &= \emptyset \\
D_4 &= \emptyset
\end{aligned}
$$

Estimated distance of state 3 is given by

$$\delta_1(\neg A \wedge \neg B \wedge C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G) = 3$$



# Distance estimation: example II, distance 1 to 3






# Distance estimation: example 2, distance 1 to 3

$$
\begin{aligned}
D_0 &= \{\neg A, \neg B, C\} \\
D_1 &= \emptyset \\
D_2 &= \emptyset
\end{aligned}
$$

Estimated distance of state 3 is given by

$$\delta_1(\neg A \wedge B \wedge C) = 1$$

In fact, all states have estimated distance $\leq 1$ from state 1.

CONCLUSION: Accuracy of distance estimates very much depends on the choice of state variables.



# PDDL: domain files

A domain file consists of

- (define (domain DOMAINNAME)
- a :requirements definition (use :adl :typing by default)
- definitions of types (each parameter has a type)
- definitions of predicates
- definitions of operators

## Example: blocks world in PDDL

```
(define (domain BLOCKS)
  (:requirements :adl :typing)
  (:types block - object
          blueblock smallblock - block)
  (:predicates (on ?x - smallblock ?y - block)
               (ontable ?x - block)
               (clear ?x - block)
               )
```



## PDDL: operator definition

- (:action OPERATORNAME
- list of parameters: (?x - type1 ?y - type2 ?z - type3)
- precondition: a formula

```
<schematic-state-var>
(and <formula> ... <formula>)
(or <formula> ... <formula>)
(not <formula>)
(forall (?x1 - type1 ... ?xn - typen) <formula>)
(exists (?x1 - type1 ... ?xn - typen) <formula>)
```



- effect:

```
<schematic-state-var>
(not <schematic-state-var>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)
```



```
(:action fromtable
    :parameters (?x - smallblock ?y - block)
    :precondition (and (not (= ?x ?y))
                       (clear ?x)
                       (ontable ?x)
                       (clear ?y))
    :effect
      (and (not (ontable ?x))
           (not (clear ?y))
           (on ?x ?y)))
```

## PDDL: problem files

A problem file consists of

- (define (problem PROBLEMNAME)
- declaration of which domain is needed for this problem
- definitions of objects belonging to each type
- definition of the initial state (list of state variables initially true)
- definition of goal states (a formula like operator precondition)



```
(define (problem blocks-10-0)
  (:domain BLOCKS)
  (:objects a b c - smallblock)
            d e - block
            f - blueblock)
  (:init (clear a) (clear b) (clear c) (clear d) (cle
         (ontable a) (ontable b) (ontable c)
         (ontable d) (ontable e) (ontable f))

  (:goal (and (on a d) (on b e) (on c f)))
)
```



## Example run on the FF planner

```
edu/PS04> ./ff -o hamiltonian.pddl -f ham1.pddl
ff: parsing domain file, domain 'HAMILTONIAN-CYCLE' (
ff: parsing problem file, problem 'HAM-1' defined
ff: found legal plan as follows
step    0: GO A B
        1: GO B D
        2: GO D F
        3: GO F C
        4: GO C E
        5: GO E A
0.01 seconds total time
```