Conformant Planning

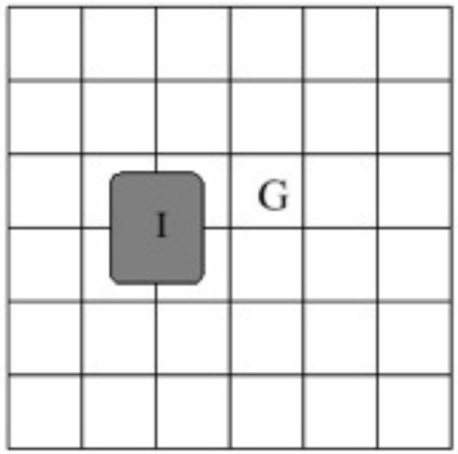
First Step toward Planning Under Uncertainty

Date

Conformant Planning

- Basic assumption in classical planning: the initial state is fully known
- What if we don't know everything about the initial state?
- Conformant planning -- like classical planning, but instead of a single possible initial state, a set of possible initial states
- Other forms of uncertainty:
  - Uncertainty about the effect of actions (non-deterministic, stochastic)
  - Some conformant planning algorithms can deal with non-deterministic effects
- Related issues:
  - Observability: can we observe information about the current state?
  - Conformant planning: no observations during plan execution

Conformant Planning: the Trouble with Incomplete Info



Problem: A robot must move from an **uncertain** I into G with **certainty**, one cell at a time, in a grid nxn

- Conformant and classical planning look similar except for uncertain I (assuming actions are deterministic).
- Yet plans may be quite different: **best conformant plan** above **must move robot to a corner first!** (in order to localize)

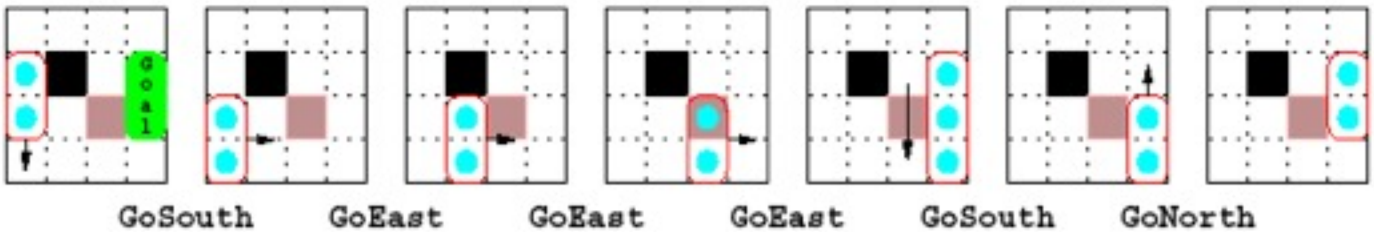
Conformant Planning

- Conformant Planning problem $\langle P,A,I,G \rangle$
  - I is an arbitrary formula, and any state s that satisfies I is a possible initial state
  - A can be non-deterministic. Later we will focus on deterministic effects
- Model -- identical to classical planning (possibly non-deterministic) automaton with multiple initial states.
- Solution -- a plan that is guaranteed to take us from any initial state to some goal state, no matter what the effect of actions is.
- Language -- like strips except:
  - Initial state described by a formula -- any assignment satisfying it is a legal state
  - Non-determinism can be captured by disjunctive effects: p v -p

Belief States

- Central concept: **belief state** --- the set of possible (world) states
  - Initial belief state: $\{s \mid s \models I\}$
- If our current belief state is b and we apply action a, then we reach a new belief state $b' = \{a(s) \mid s \models b\}$

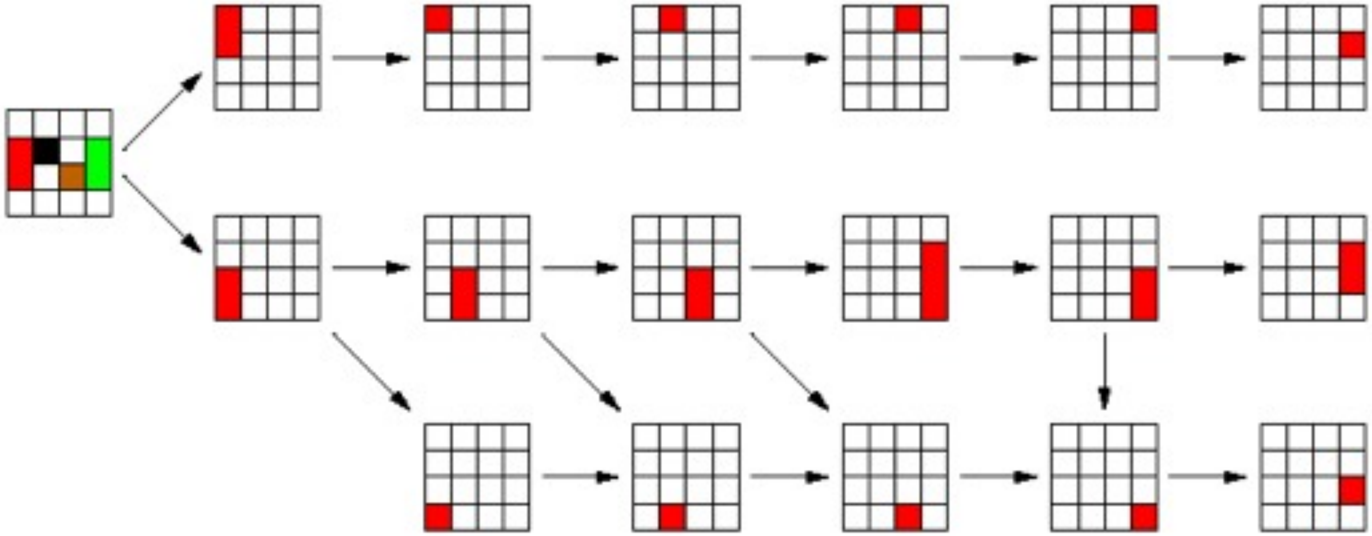
Example:



Search in Belief Space

- Conformant planning can be viewed as the problem of finding a path in belief space
  - Initial state: initial belief state
  - Goal state: any belief state b such that $s \in b \Rightarrow s \models g$
  - Actions: $a(b) = \{a(s) \mid s \models b\}$
  - In general, a belief state could require an exponentially large (in # of state variables) description

The Search Space



Remark:

- the search space is $Pow(\mathcal{S})$
- $\mathcal{S}$ contains 15 states,
- $Pow(\mathcal{S})$ contains 32767 belief states!

Complexity

- We can verify that a classical plan is true in time linear in plan length and # of propositions
- Verifying that a conformant plan is correct may be intractable
  - Initial state: initial belief state
  - Goal state: any belief state b such that $s \in b \Rightarrow s \models g$
  - Actions: $a(b)=\{a(s) \mid s \models b\}$
  - In general, a belief state could require an exponentially large (in # of state variables) description

Generating Conformant Plans

- Two main issues:
  - How do we represent belief states efficiently?
    - Small size desirable
    - Ability to quickly detect goal satisfaction
    - Ability to quickly detect which action is applicable
  - How can we generate good heuristic estimates?

Special Case

- Standard STRIPS actions
- Initial state: the value of some propositions is known, the value of others is completely unknown (no constraints of the form p v q)
- Solution:???

Representing Belief States

1. Explicit representation: Maintain a set of states
   - All operations require time linear in number of possible states
   - All operations are conceptually simple
   - The number of possible states can be very large
   - Does not work in practice
2. Symbolic representation: Maintain formula ϕ over state propositions
   - s is a possible state iff it satisfies ϕ
   - Key issue: how do we represent ϕ
     - Different choices affect the computational and conceptual difficulty of different operations (update, verification of goal/preconditions) and the size of the formula

Alternative Symbolic Representations

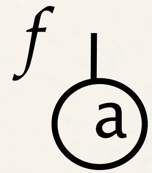
- Logical formula w/o constraints
- Conjunctive Normal Form: Conjunction of Disjunctions
    - (pvqvr) & (-pvwvd) & (-wvqvs)
    - Checking whether a precondition/goal holds require solving un-sat problem
- Disjunctive Normal Form: Disjunction of Conjunctions
    - (p&q&r) v (-p&w&d) v (-w&qs&)
    - Checking whether a condition holds is easy
    - The number of conjuncts can grow rapidly
- Binary Decision Diagrams

Binary Decision Diagrams

- A data structure used for compactly representing boolean functions
- Made popular by work on program verification
- Based on recursive Shannon expansion
- $$f = x f_x + x' f_{x'}$$
- Canonical representation
  - reduced ordered BDDs (ROBDD) are canonical (= there is only one way to represent any function given a fixed variable order)

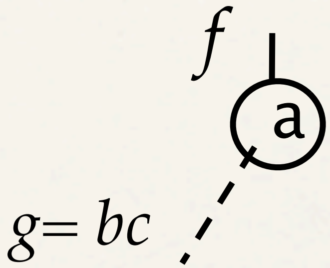
Recursive Shannon Expansion for f= ac + bc

Recursive Shannon Expansion for f= ac + bc



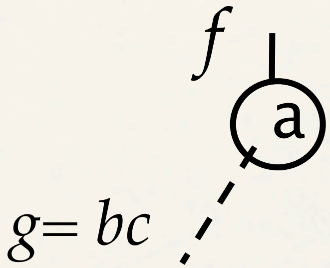
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$



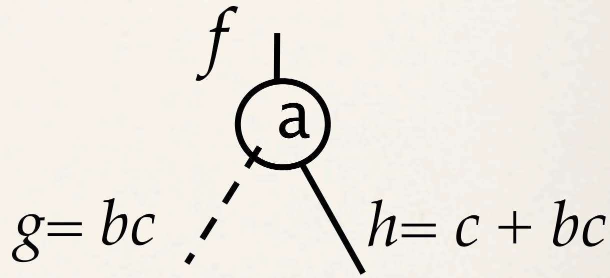
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$



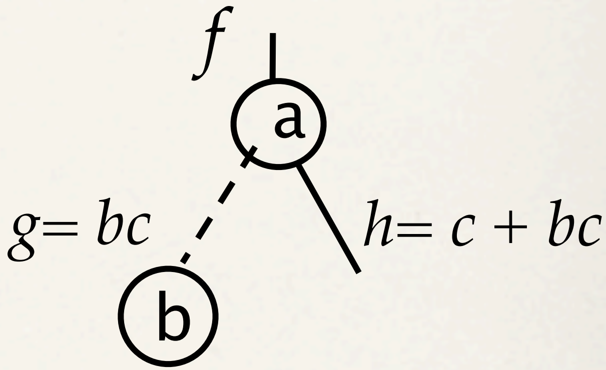
Recursive Shannon Expansion for f= ac + bc

- $f = ac + bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$



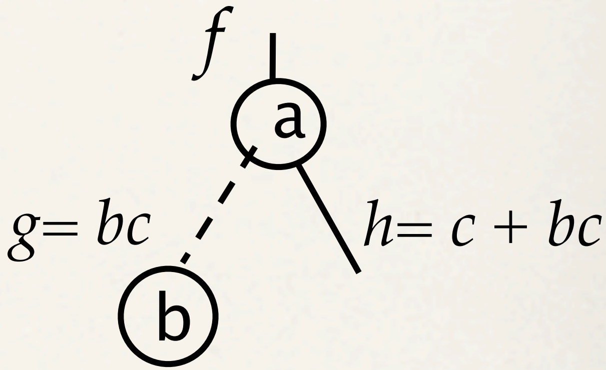
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$



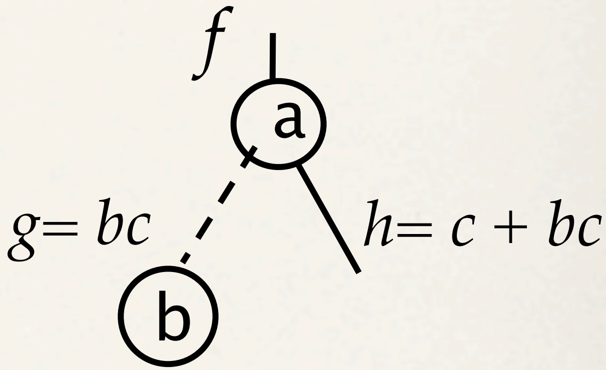
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$
  - $g_b = (bc)_{|b=1} = c$



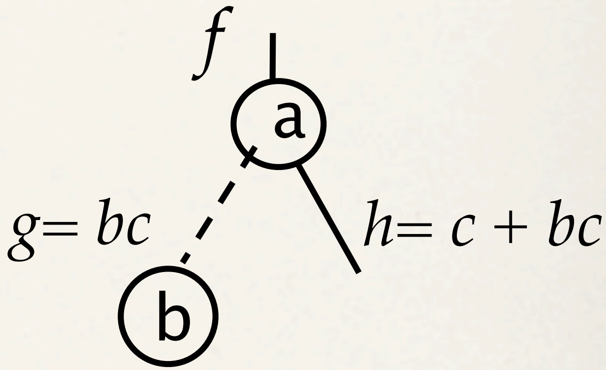
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$
  - $g_b = (bc)_{|b=1} = c$
  - $g_{b'} = (bc)_{|b=0} = 0$



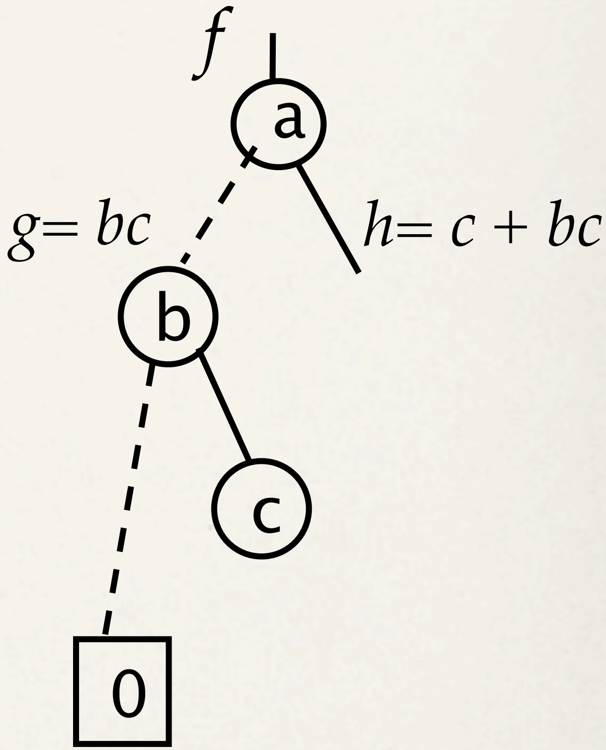
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$
  - $g_b = (bc)_{|\,b=1} = c$
  - $g_{b'} = (bc)_{|\,b=0} = 0$
  - $h_b = (c+bc)_{|\,b=1} = c$



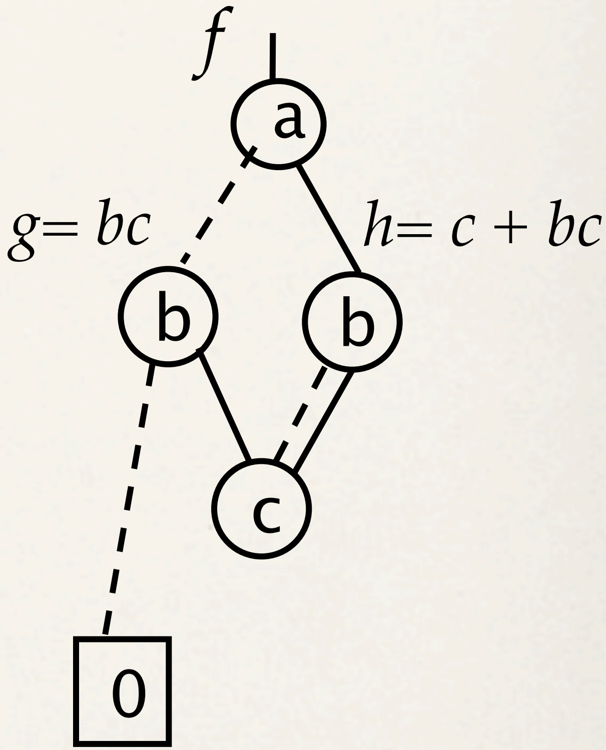
Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$
  - $g_b = (bc)_{|b=1} = c$
  - $g_{b'} = (bc)_{|b=0} = 0$
  - $h_b = (c+bc)_{|b=1} = c$
  - $h_{b'} = (c+bc)_{|b=0} = c$



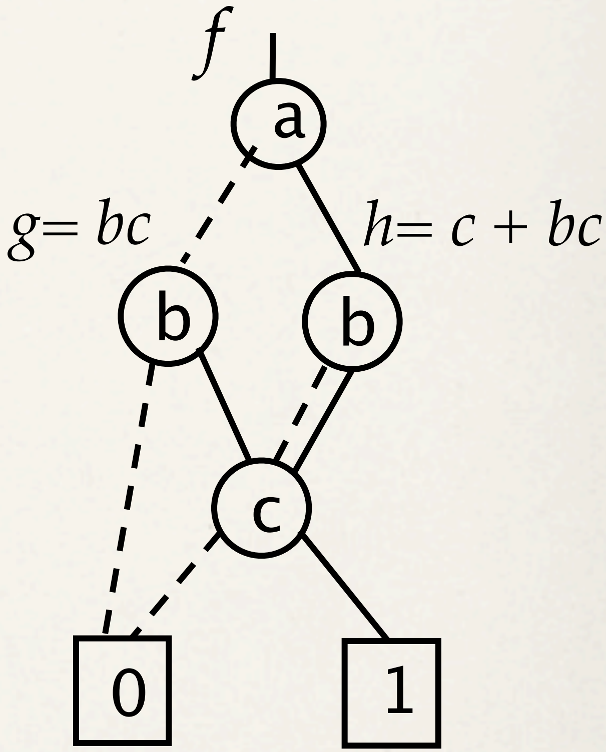
Recursive Shannon Expansion for f= ac + bc

- $f = ac + bc$
  - $h = f_a = f(a{=}1) = c + bc$
  - $g = f_{a'} = f(a{=}0) = bc$
  - $g_b = (bc)_{|b=1} = c$
  - $g_{b'} = (bc)_{|b=0} = 0$
  - $h_b = (c+bc)_{|b=1} = c$
  - $h_{b'} = (c+bc)_{|b=0} = c$



Recursive Shannon Expansion for f= ac + bc

- $f = ac+bc$
  - $h = f_a = f(a=1) = c + bc$
  - $g = f_{a'} = f(a=0) = bc$
  - $g_b = (bc)_{\mid b=1} = c$
  - $g_{b'} = (bc)_{\mid b=0} = 0$
  - $h_b = (c+bc)_{\mid b=1} = c$
  - $h_{b'} = (c+bc)_{\mid b=0} = c$

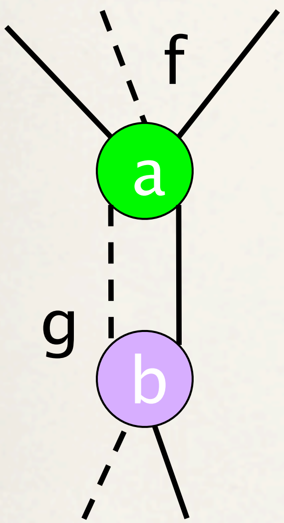


BDD operations

- When the two outgoing edges of a node point to the same node, remove it
-

BDD operations

- When the two outgoing edges of a node point to the same node, remove it

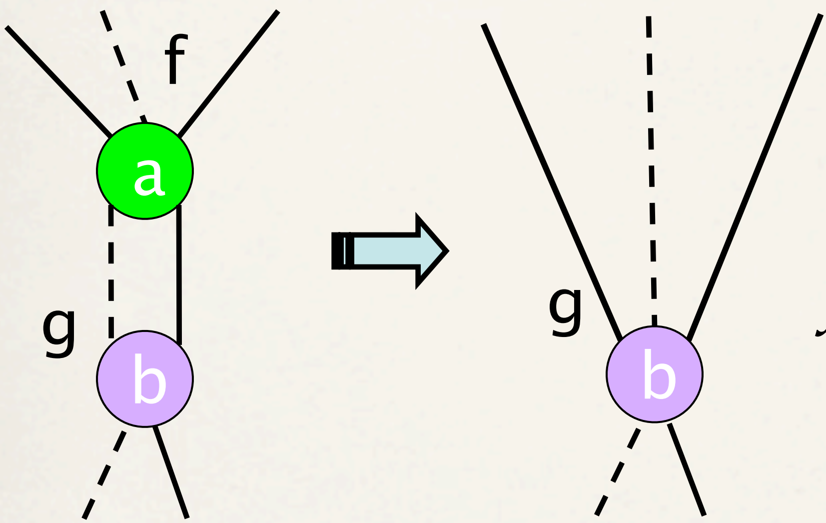


$$f = a'\,g(b) + a\,g(b) = g(b)$$
$$(f_a + f_{a'} = 1)$$

-

BDD operations

- When the two outgoing edges of a node point to the same node, remove it



$$f = a'\,g(b) + a\,g(b) = g(b)$$

$$(f_a + f_{a'} = 1)$$

-

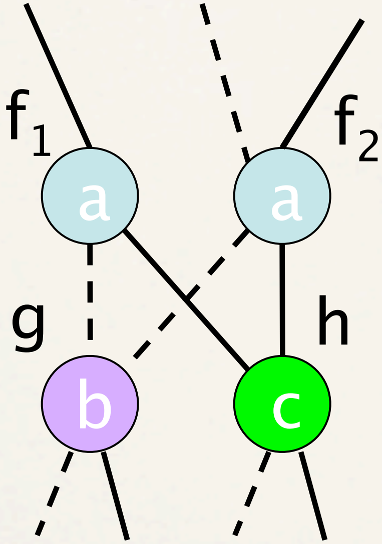
BDD Operations

- Merge duplicate nodes

-

BDD Operations

- Merge duplicate nodes

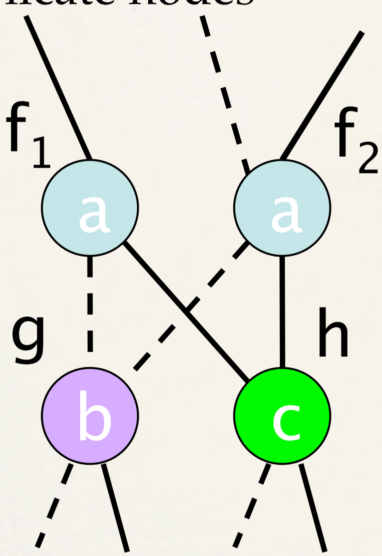


$$f_1 = a'\, g(b) + a\, h(c) = f_2$$

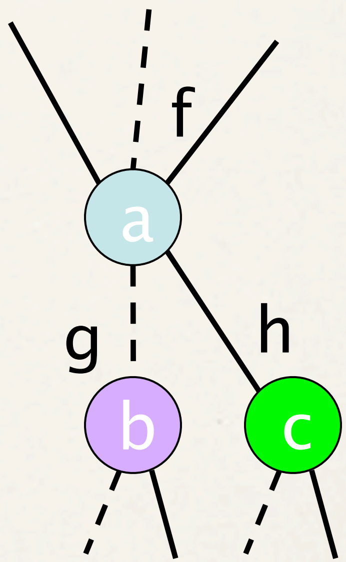
-

BDD Operations

- Merge duplicate nodes



$$f_1 = a'\, g(b) + a\, h(c) = f_2$$



$$f = f_1 = f_2$$

-

BDD Construction

- You can start with a decision tree and merge: example f=ac+bc
- Reduced, ordered, BDD:
  - Reduced -- no additional reductions can be applied
  - Ordered -- the order of variables in a path from the root to a leaf is fixed

BDD Construction

- You can start with a decision tree and merge: example f=ac+bc

Truth table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0

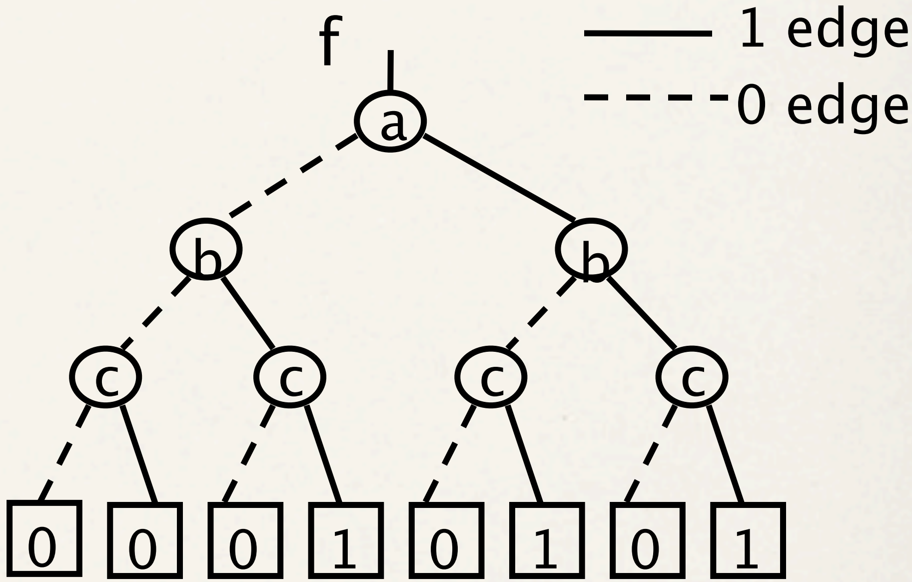
- Reduced, ordered, BDD:
  - Reduced -- no additional reductions can be applied
  - Ordered -- the order of variables in a path from the root to a leaf is fixed

BDD Construction

- You can start with a decision tree and merge: example f=ac+bc

Truth table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0

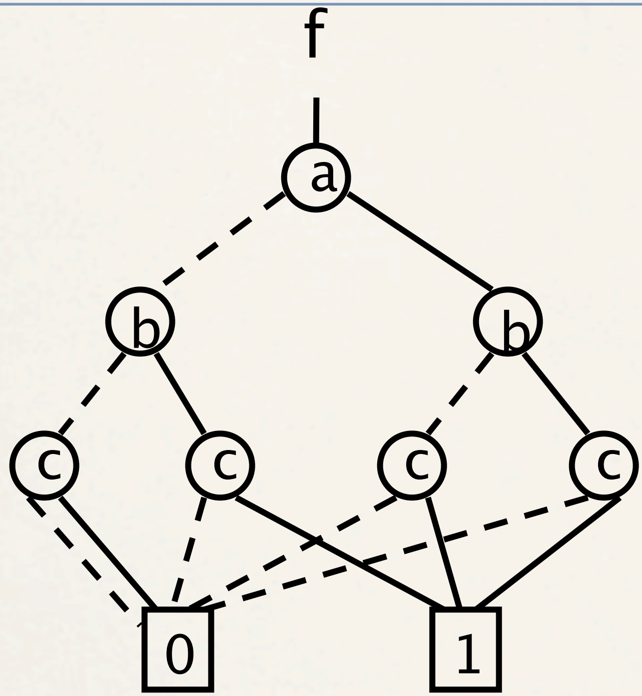


Decision tree

- Reduced, ordered, BDD:
  - Reduced -- no additional reductions can be applied
  - Ordered -- the order of variables in a path from the root to a leaf is fixed

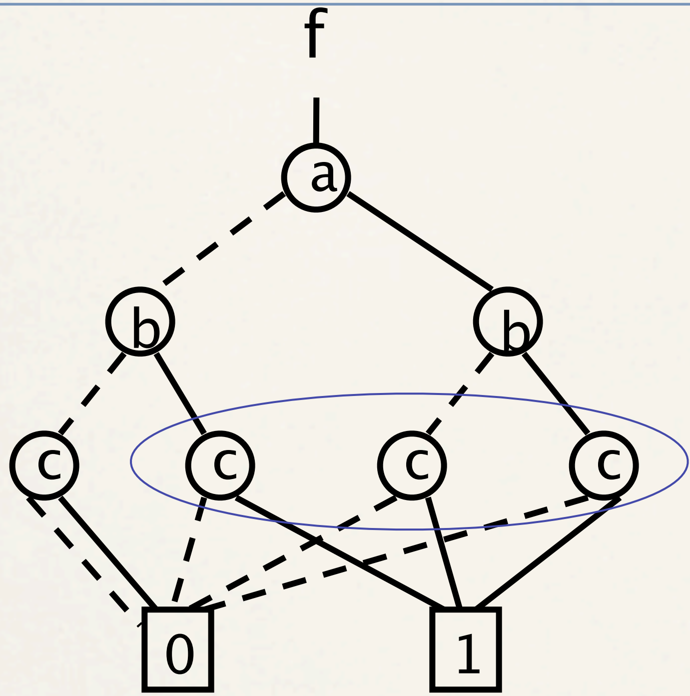
BDD Construction (continued)

BDD Construction (continued)



1. Merge terminal nodes

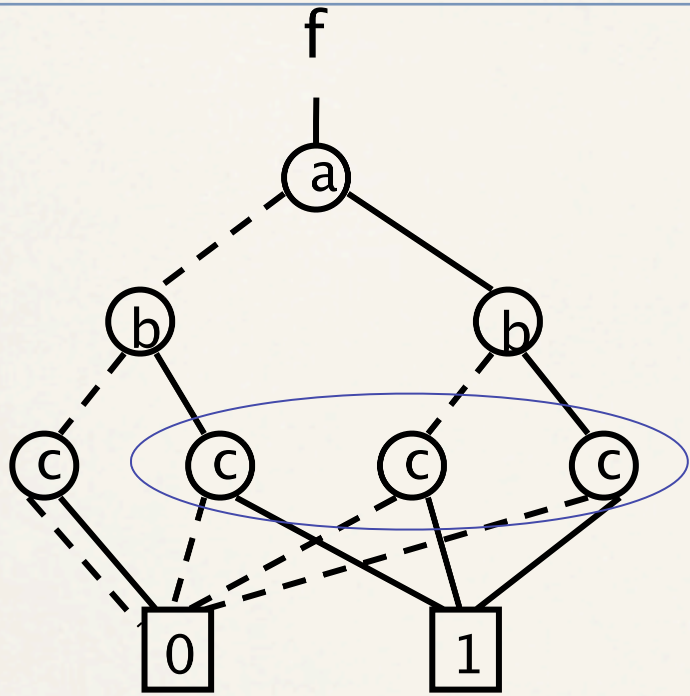
BDD Construction (continued)



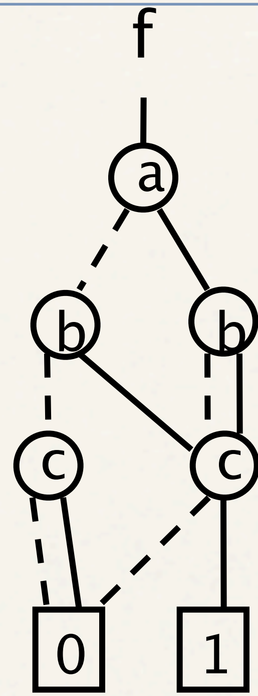
1. Merge terminal nodes

2. Merge duplicate nodes

BDD Construction (continued)

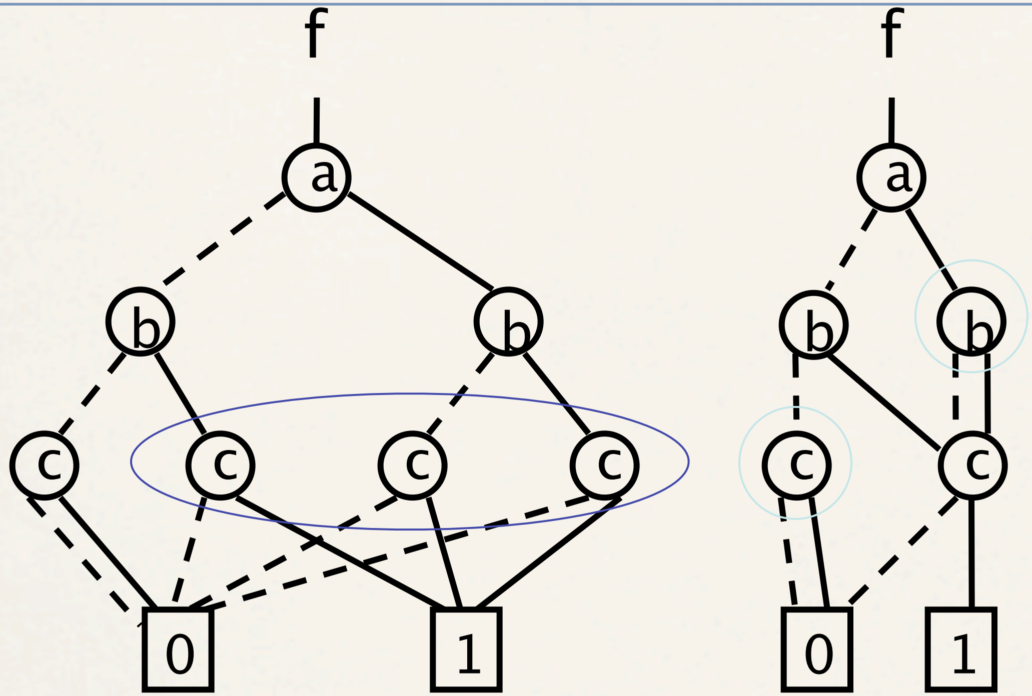


1. Merge terminal nodes



2. Merge duplicate nodes

BDD Construction (continued)

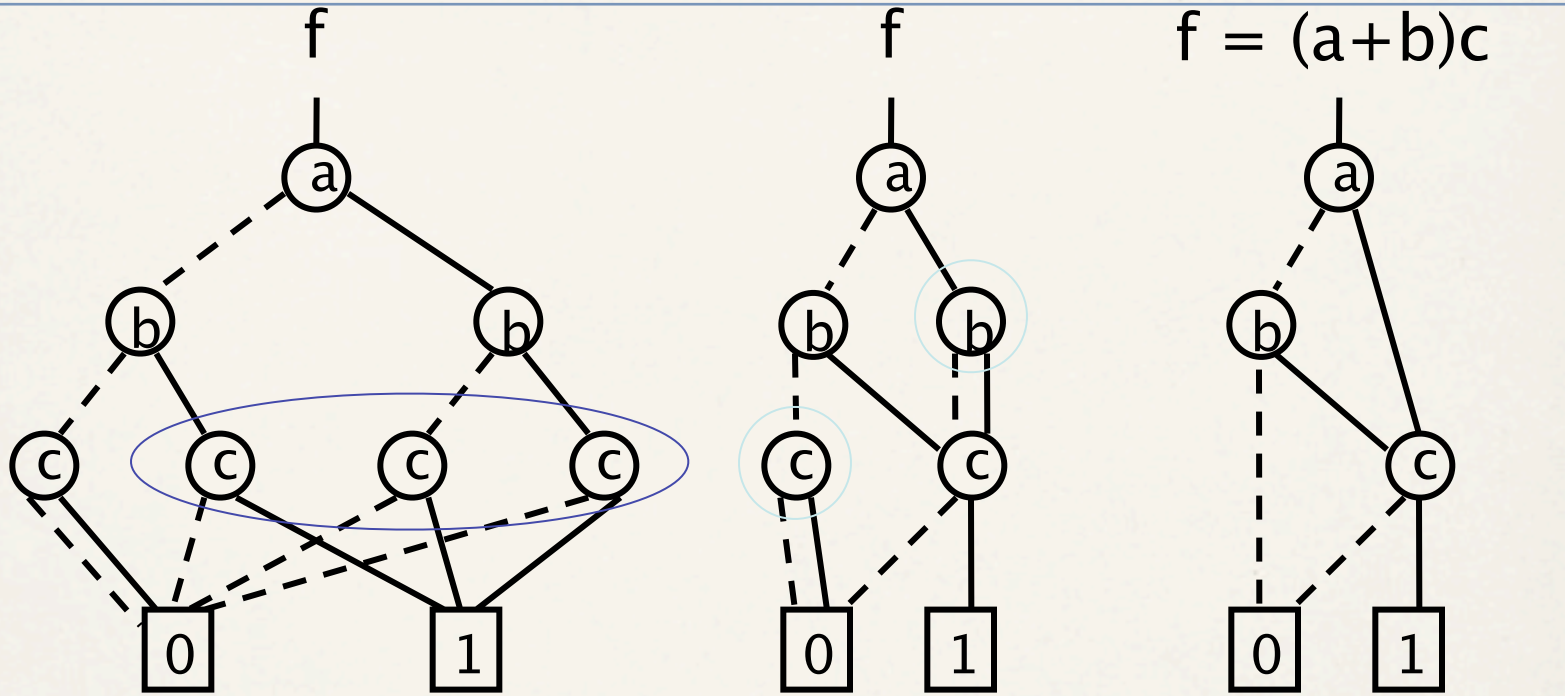


1. Merge terminal nodes

2. Merge duplicate nodes

3. Remove redundant nodes

BDD Construction (continued)



1. Merge terminal nodes

2. Merge duplicate nodes

3. Remove redundant nodes

BDDs Support Efficient Logical Manipulations

- Negating a function (very simple??)
- Conjoining two functions
- Disjoining two functions
- Others
- Operations utilize the recursive definition of the function

Implicit Representation

- This is also a representation via a formula, but with different propositions
- Essentially, this is the same formula generated by a SAT-encoding
- A state s is possible currently if there is a satisfying assignment that assigns the propositions at time t the same values as s.
  - Update is very easy
  - Checking whether a condition holds now requires verifying that a formula is unsatisfiable
  - The formula can be simplified during run-time

Searching in Belief Space

- All current planners use forward search
- Main problem: heuristics are difficult to generate
- Size heuristic: $hs(b) = -1 * \left| \{s : s \in b\} \right|$
  - Pushes toward belief states with more certainty
- That's about it ... not strong enough.

The Translation-Based Approach

- In classical planning, if we know the initial state, we know the current state simply from the description of the actions
- Basic idea: maintain a copy of each proposition for each possible initial state
    - $p/i_1, p/i_2, \ldots, p/i_k$
    - And also a "general" copy: p
- Generate actions that update all copies
    - If $p \dashrightarrow q$ is an original effect of a, add $p/i_j \dashrightarrow q/i_j$ for every $1 \le j \le k$
- This way, we know what's true now as a function of what was true initially
- We can also deduce that if p/i_j holds now for every $1 \le j \le k$, then p holds.
    - This way, we can know whether some precondition or goal condition holds
- So far, pretty wasteful because we may have exponentially many initial states

The Translation-Based Approach

- We can use this idea to generate a new classical planning problem
- Propositions: p, p/i_j for every possible proposition p and every possible initial state i_j
- Actions:
  - the original actions, with effects modified as described before
  - special inference actions: $p/i_1 \wedge p/i_2 \wedge \ldots \wedge p/i_k \rightarrow p$ for every proposition p
- Initial state: p/i_j is true iff p holds in possible initial state i_j
- Goal state: g (as in the original problem)
- We get a classical planning problem, and we can solve it with a classical planner
- No need for special heuristics!

The Translation-Based Approach

- Actually, in the literature:
- Propositions: Kp, Kp/i_j is used
  - Kp -- p is known
  - Kp/i_j -- p is known given i_j
  - More generally: Kp/t -- p is known given some condition t on the initial state
  - K is used in logics of knowledge: Something is known if it holds in all possible states.
  - This is captured by: $Kp/i_1 \wedge Kp/i_2 \wedge \ldots \wedge Kp/i_k \rightarrow Kp$
- The planner is reasoning about our state of knowledge

The Translation-Based Approach

- Main problem: many possible initial states
- Possible solution: use tags (conditions) that are more general
- This is not always possible, but in many problem it works
  - When it doesn't work, we're in trouble -- why?
- Example: two variables: p1,p2,...,pk. Both unknown initially.
  - 2^k possible initial states
  - Suppose that the goal is p1&...&pk, and a_i has a conditional effect: $-p_i$ --> p_i
  - According to previous slides, we need 2^k possible tags
  - We can work with 2*k tags -- one for each value of each variable
  - Reason -- the effect on tags is independent