Stavros Vassos, University of Athens, Greece stavrosv@di.uoa.gr May 2012

INTRODUCTION TO AI STRIPS PLANNING

.. and Applications to Video-games!

Course overview

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS

STRIPS planning

- What we have seen so far
  - The STRIPS formalism for specifying planning problems
  - Solving planning problems using state-based search
  - Progression planning
  - Simple heuristics for progression planning

- Can we take advantage of the information that action schemas hold to do better?

Planning graphs

- Action schemas provide useful information about the **interaction** between actions
- E.g., action A cannot take place right after B because A cancels a precondition of B
- There are many more (and more complex) conditions that would be valuable to identify!

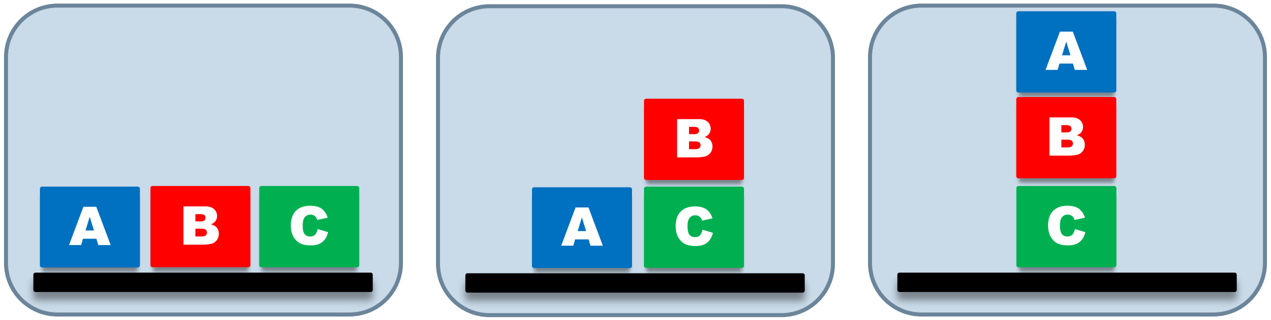
Course overview

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: **Planning graphs**, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS

Planning graphs

- Planning graph
  - Special data structure
  - Consists of a sequence of **levels**
  - Stores the effects of **all applicable actions** at every level as if they were all happening **concurrently**
  - Stores some basic **mutual exclusion** constraints between actions and literals

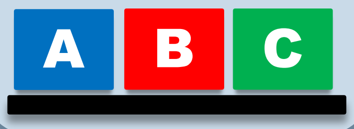
Planning graphs



On(A,Table) On(B,Table) On(C,Table) Clear(A) Clear(B) Clear(C)	On(A,Table) On(B,C) On(C,Table) Clear(A) Clear(B) Clear(Table)	On(A,B) On(B,C) On(C,Table) Clear(A) Clear(Table)

$$s_0 \xrightarrow{\text{Move(B,Table,C)}} s_1 \xrightarrow{\text{Move(A,Table,B)}} s_2$$

Planning graphs



???

???

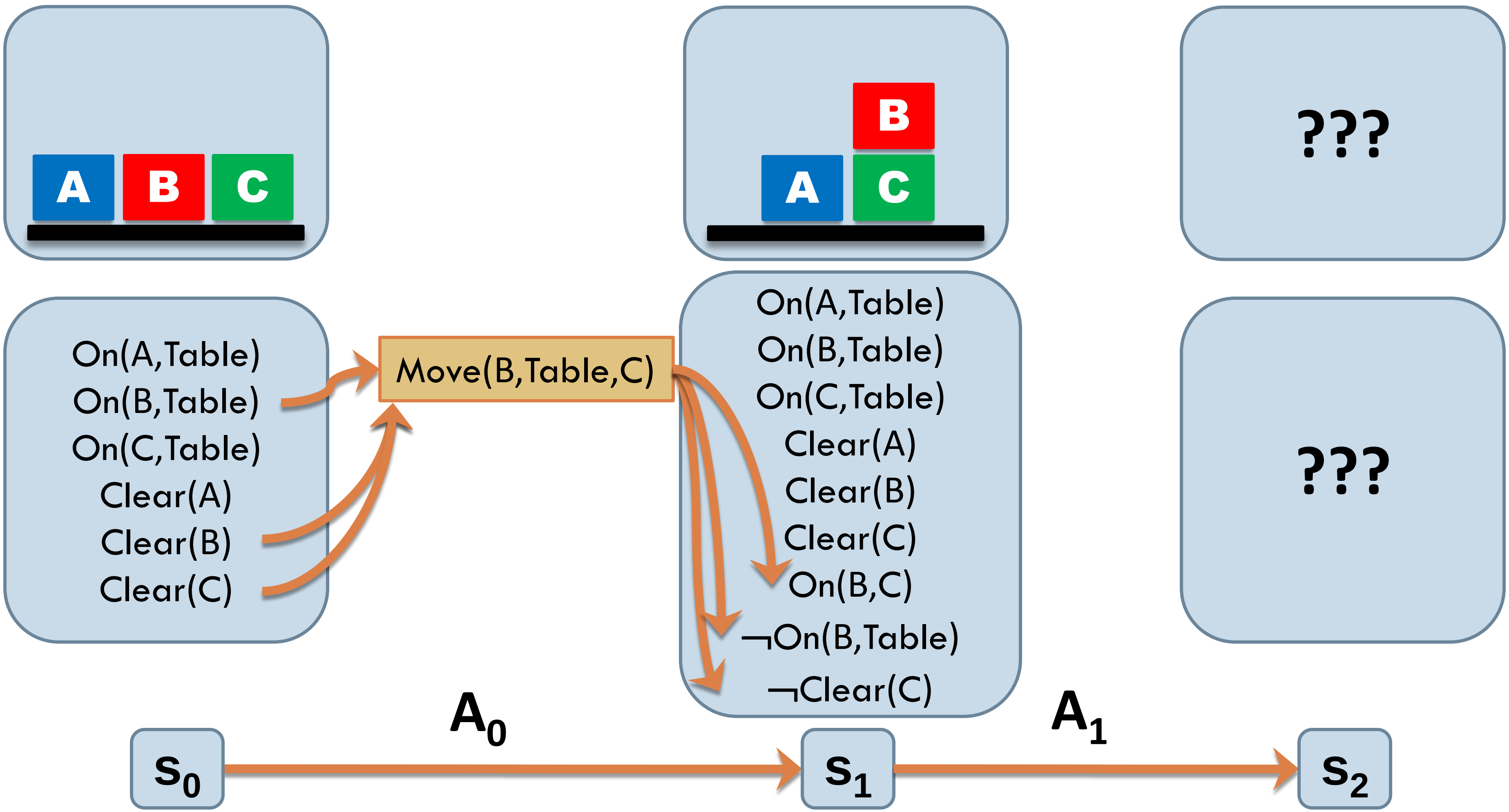
On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

???

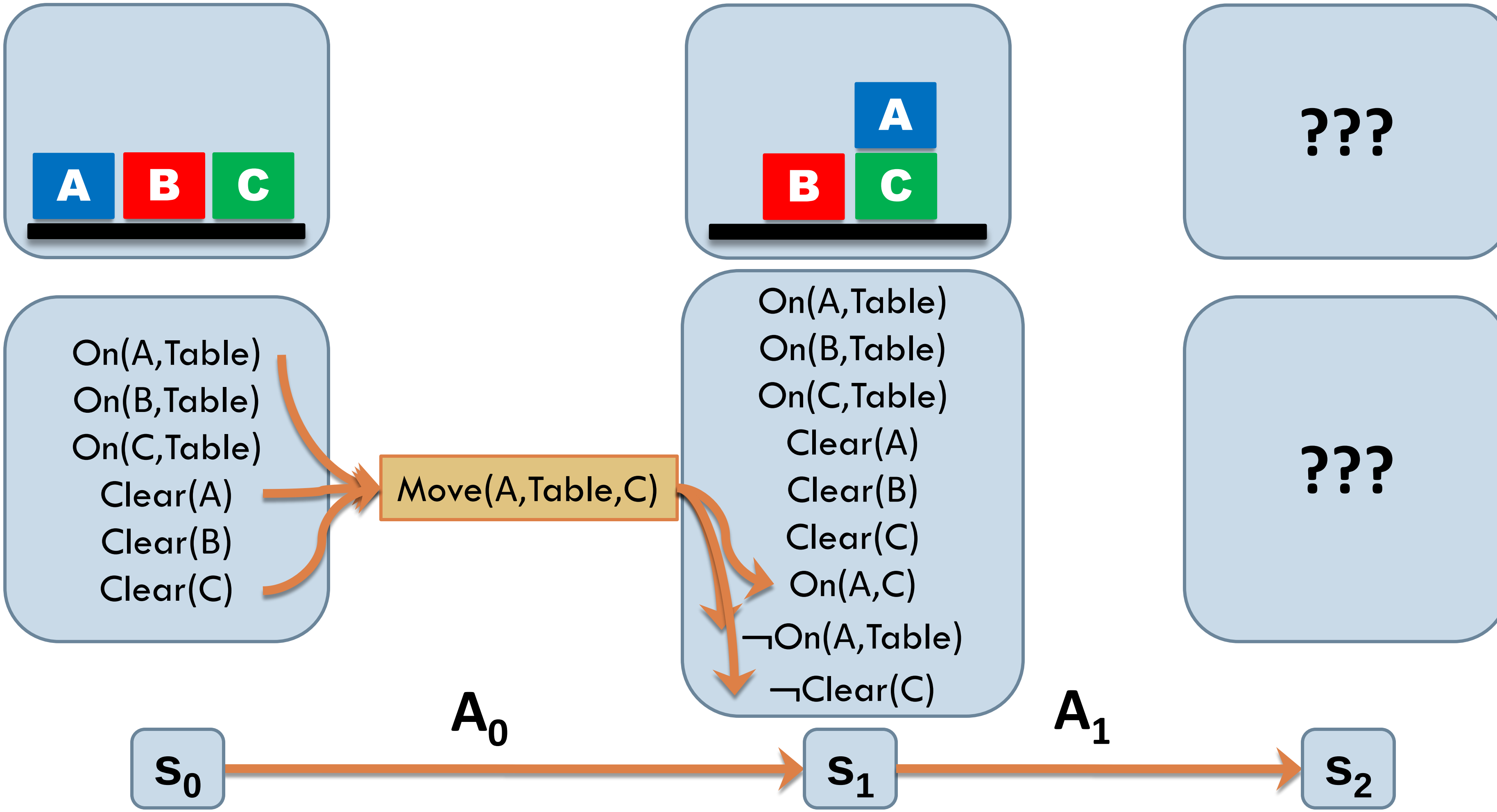
???

$S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2$

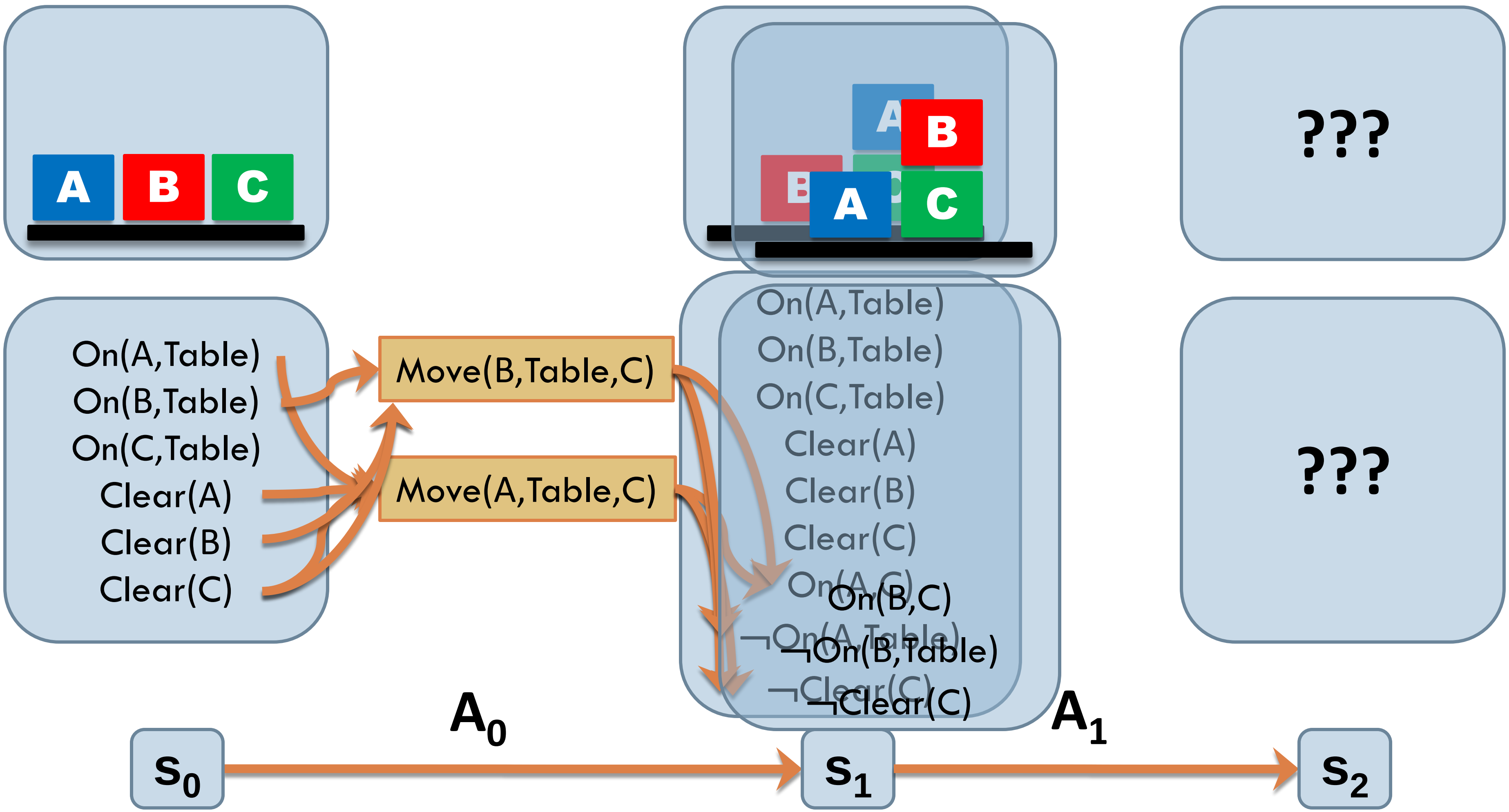
Planning graphs



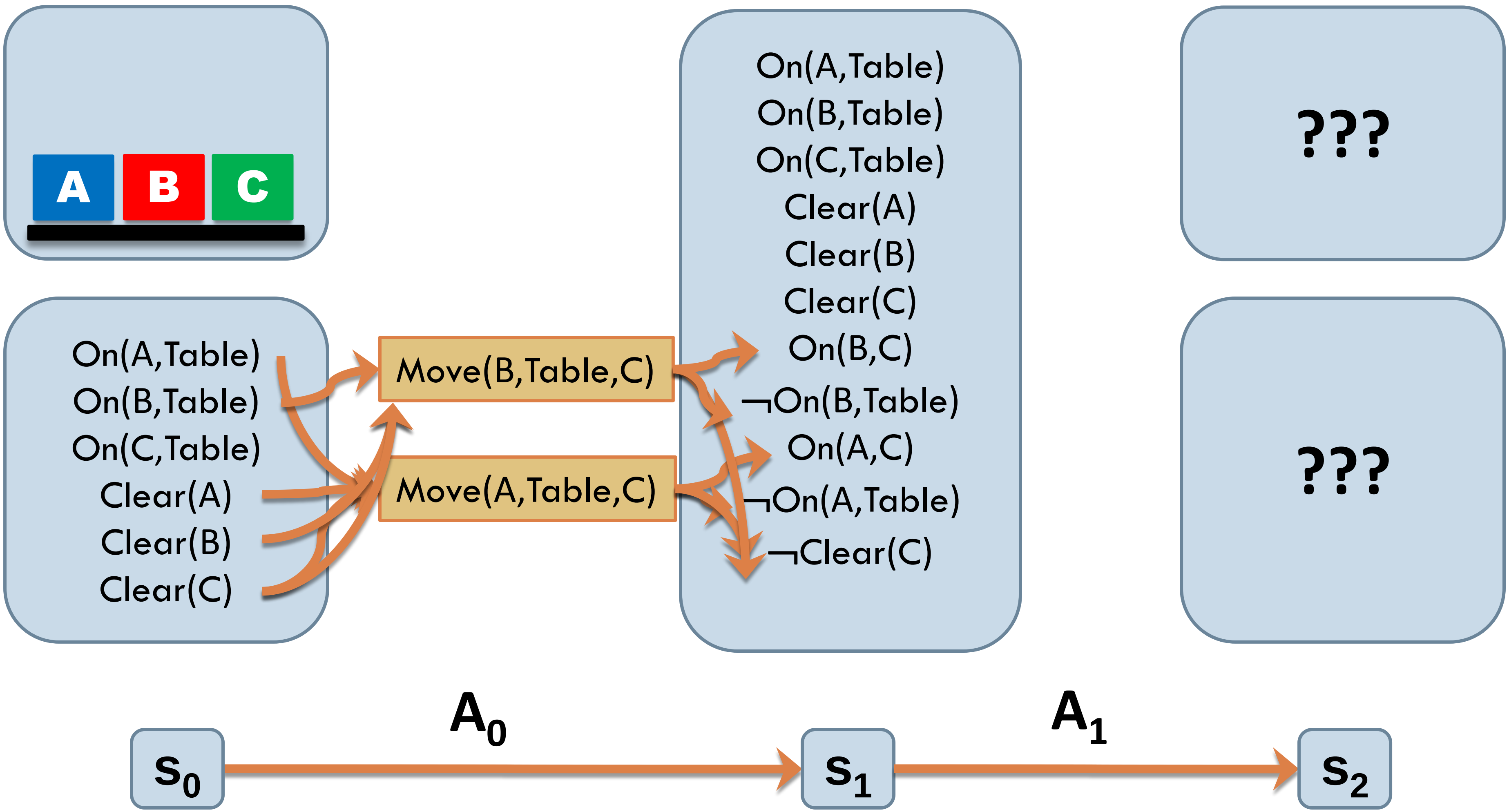
Planning graphs



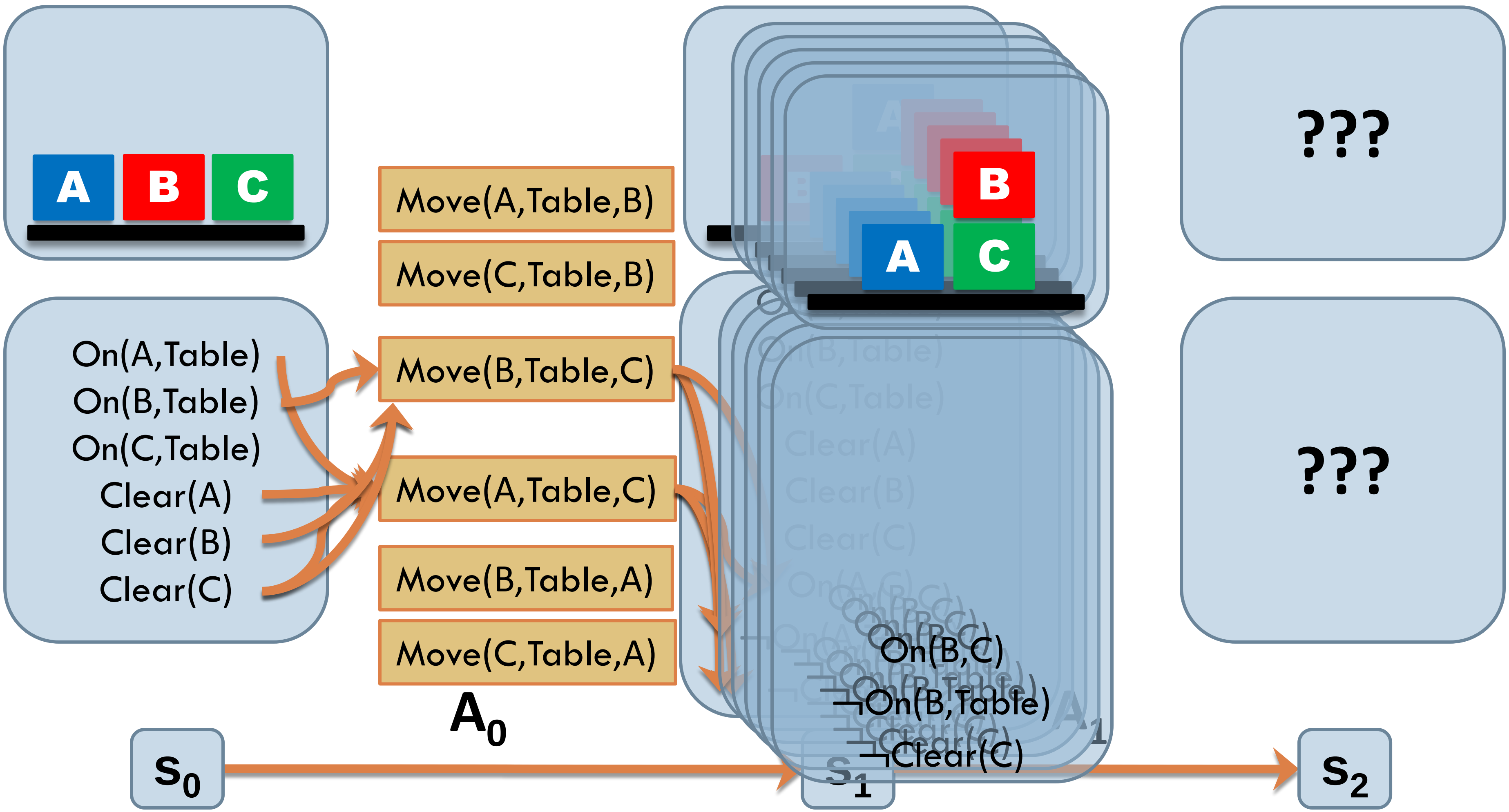
Planning graphs



Planning graphs



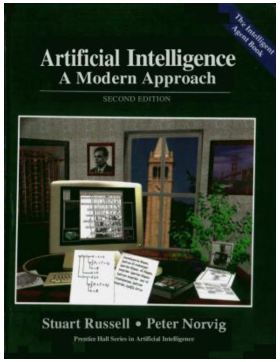
Planning graphs



Planning graphs

- Planning graph
  - Special **data structure**
  - Consists of a **sequence of levels**
  - Stores the effects of **all applicable actions** at every level as if they were all happening **concurrently**
  - Stores some basic **mutual exclusion** constraints between actions and literals

  - **.. Let's see an (even) simpler example!**



Planning graphs

- Init(Have(Cake))
- Goal(Have(Cake) $\wedge$ Eaten(Cake))
- Action(Eat(Cake)

  PRECONDITIONS: Have(Cake)

  EFFECTS: $\neg$Have(Cake) $\wedge$ Eaten(Cake))
- Action(Bake(Cake),

  PRECONDITIONS: $\neg$Have(Cake)

  EFFECTS: Have(Cake))

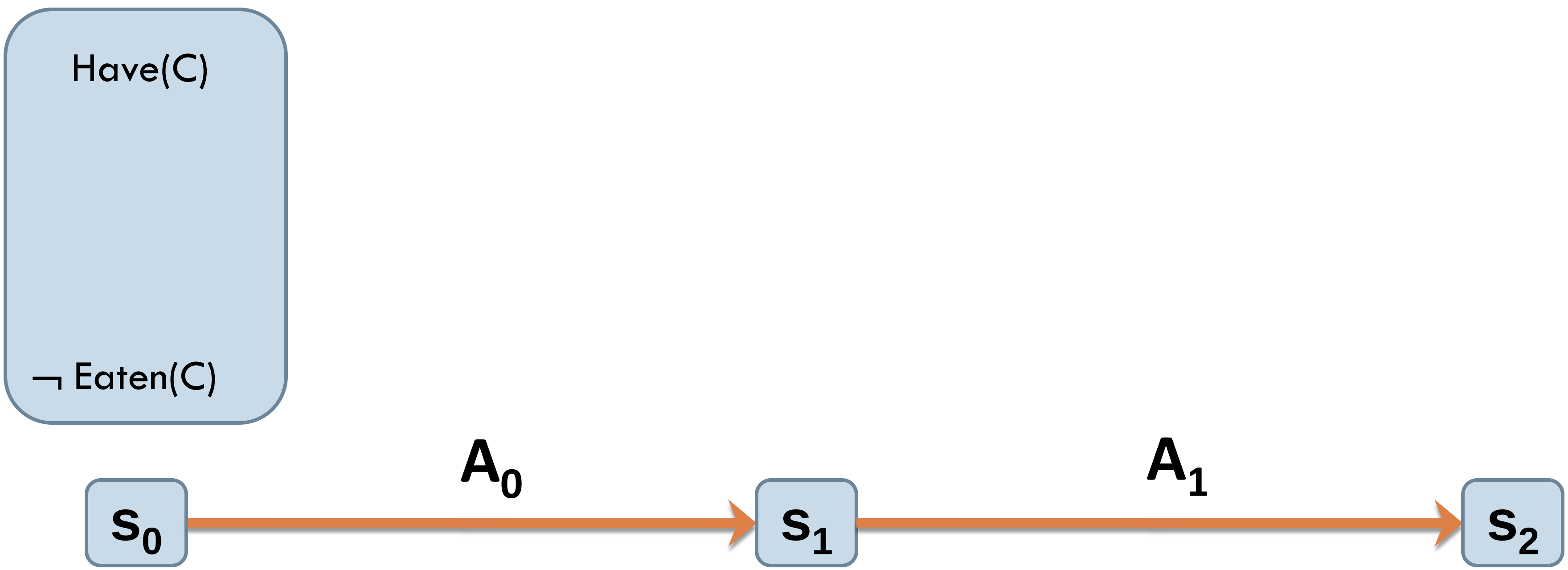
Planning graphs

- Planning graph
  - Consists of a sequence of levels that specify how the initial state is transformed under the effects of actions

  - At each level i we specify
    - A list of literals S_i
    - A list of actions A_i
    - 4 kinds of constraints or mutual exclusion links between literals in S_i and actions in A_i

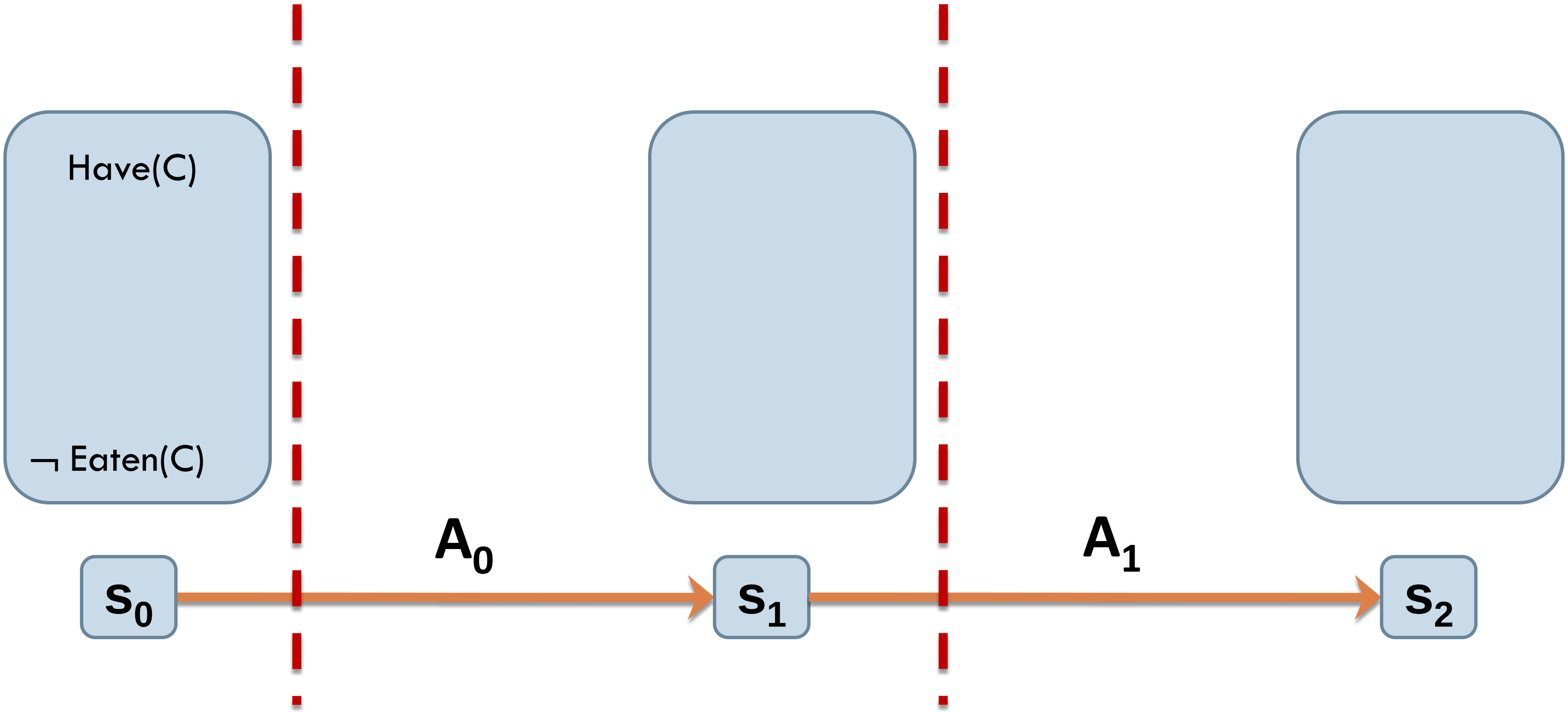
Planning graphs

- Level 0
  - S_0: the **positive literals** of the initial state **as well** as the **negative literals** implied by the closed world assumption



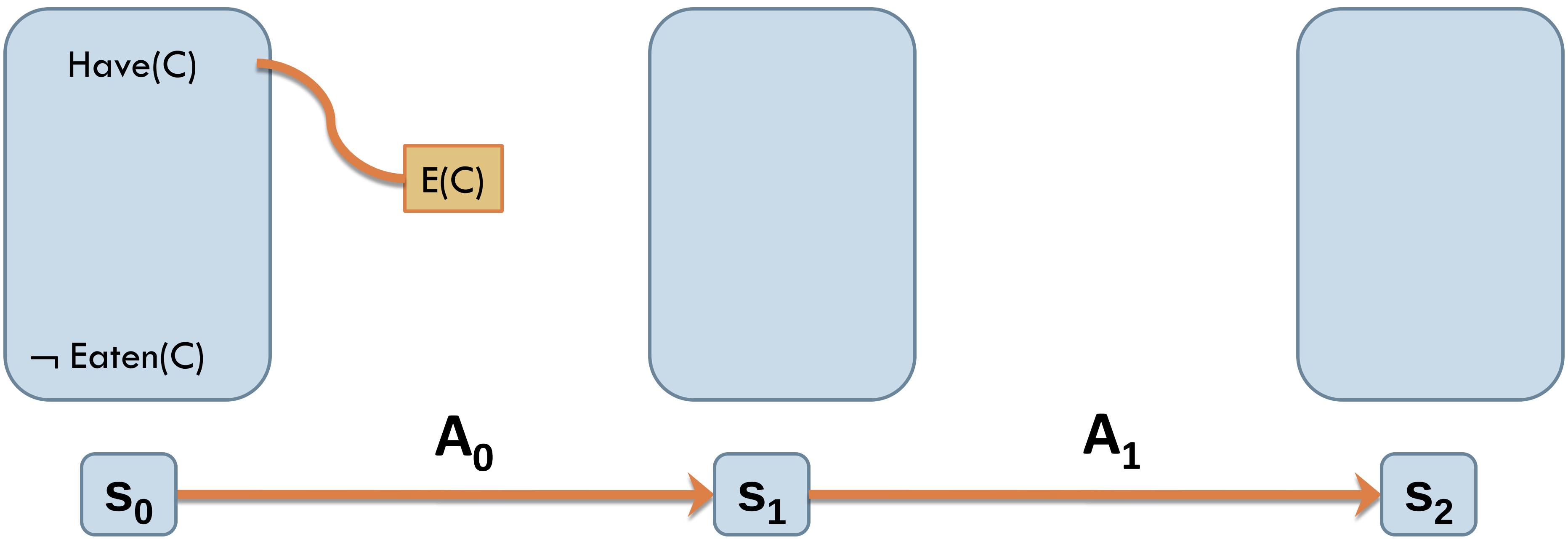
Planning graphs

- Level 0
  - We will see now how to specify $\mathbf{A_0}$ and $\mathbf{S_1}$



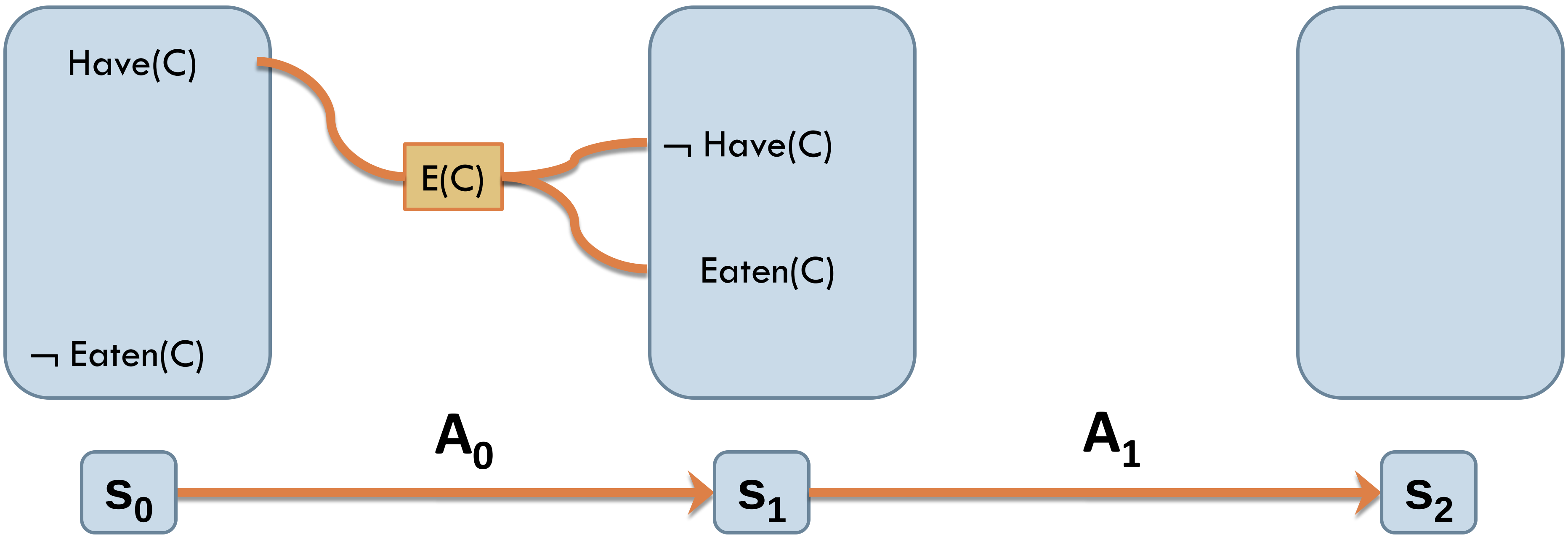
Planning graphs

- Level 0
  - A_0: the **applicable actions** in the initial state



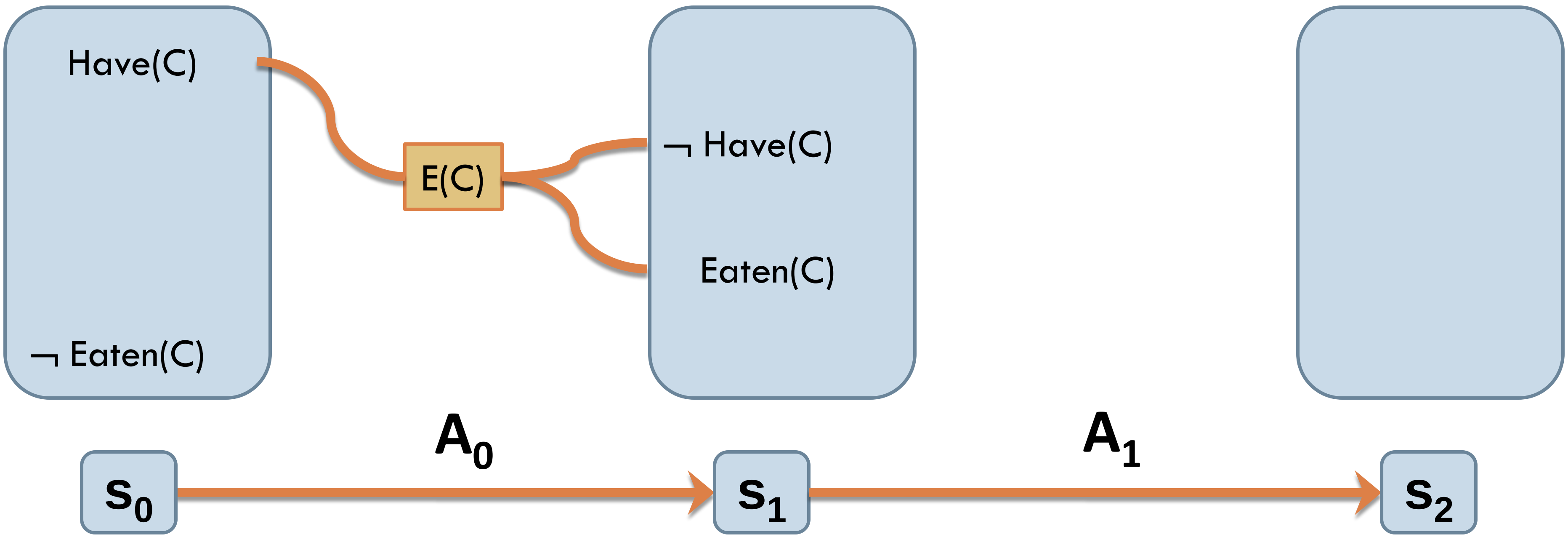
Planning graphs

- Level 0
  - S_1: the **effects** of actions that appear in A_0



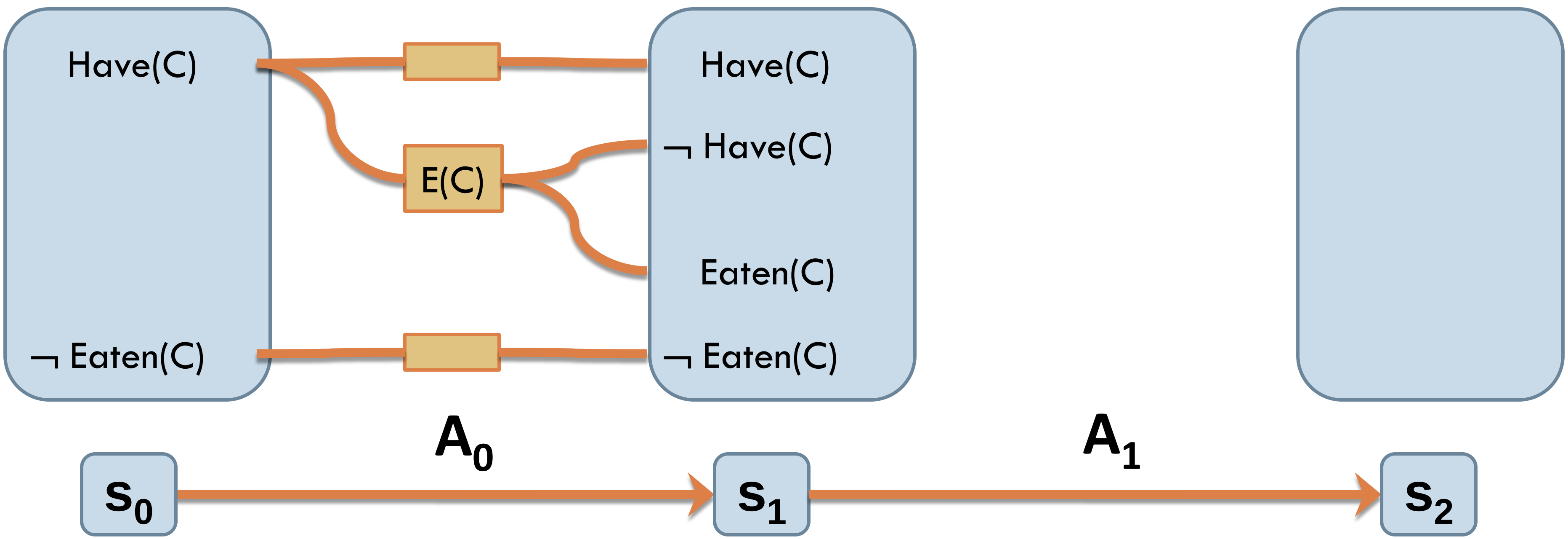
Planning graphs

- Level 0
  - We're not done yet!



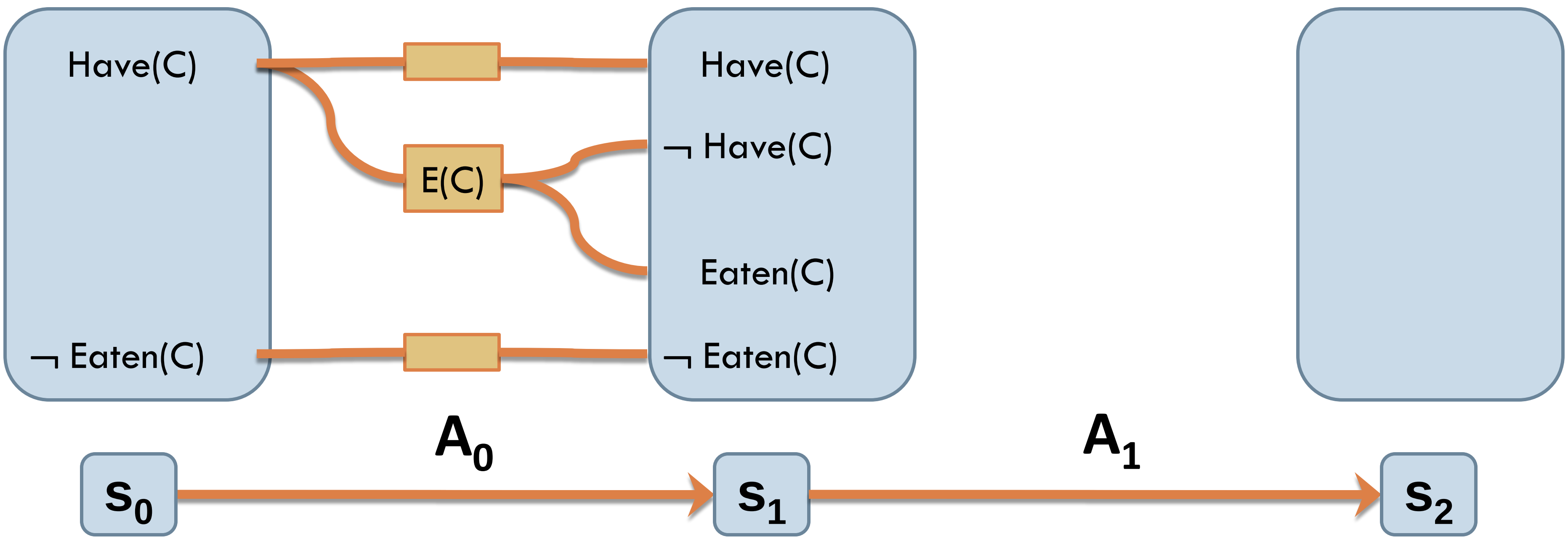
Planning graphs

- Level 0
  - We're not done yet!
  - Also add **persistence actions** that denote "inaction"



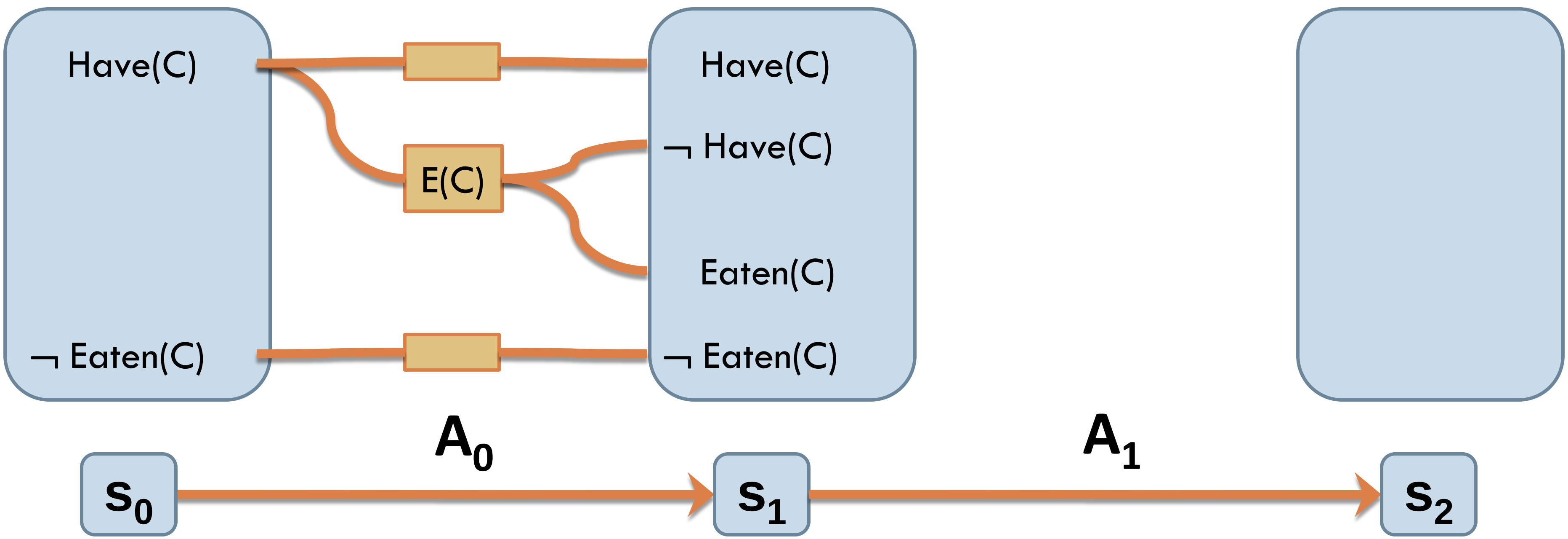
Planning graphs

- Level 0
  - A persistent action specifies that a literal **does not change truth value** between levels, e.g., here ¬ Eaten(C)



Planning graphs

- Level 0
  - We're not done yet!
  - **Mutual exclusion links**

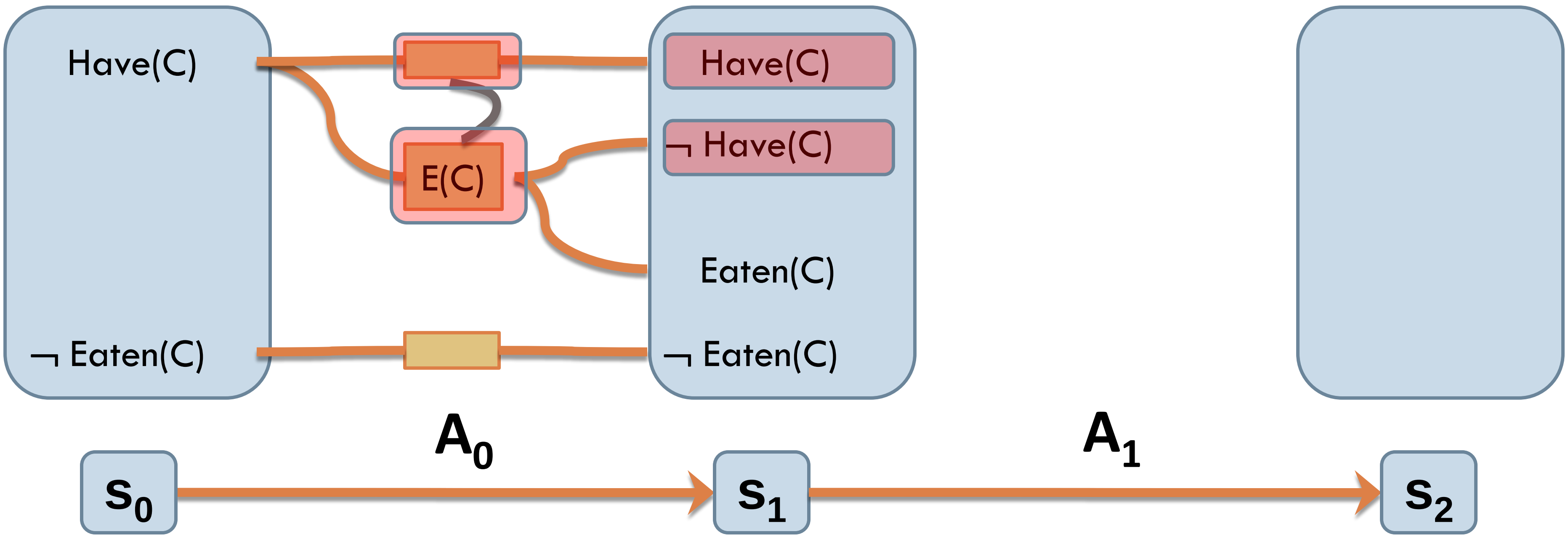


Planning graphs

- Mutual exclusion links (mutex)
  - Inconsistent effects
  - Interference
  - Inconsistent support
  - Competing needs

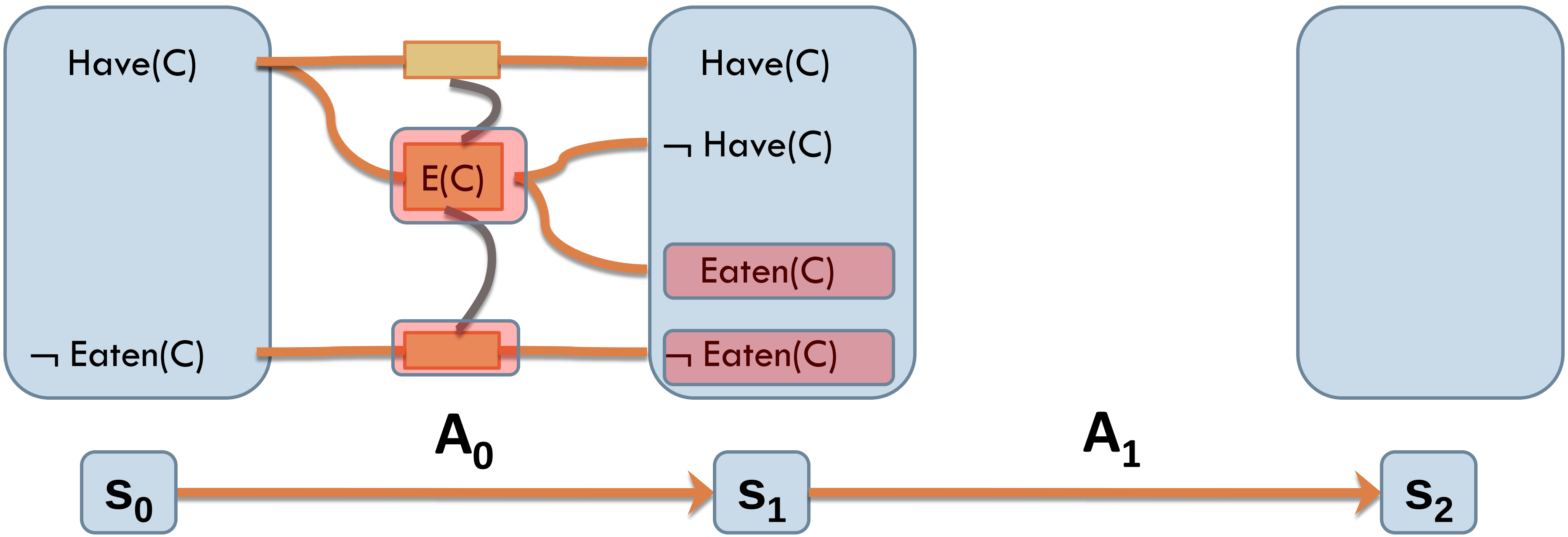
Planning graphs

- Two actions have inconsistent effects when:
  - One action cancels the effect of the other action
    - E.g., action E(C) and the persistent action for Have(Cake) have inconsistent effects



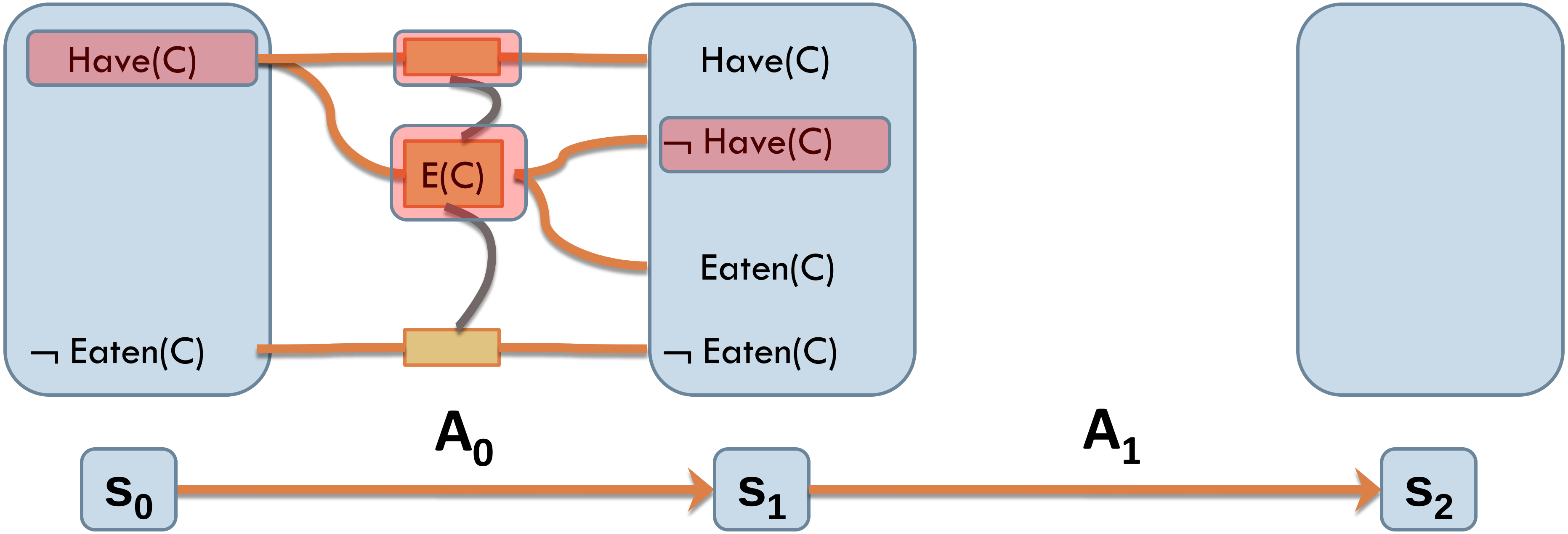
Planning graphs

- Two actions have inconsistent effects when:
  - One action cancels the effect of the other action
    - Same for action E(C) and the persistent action for ¬Eaten(C)



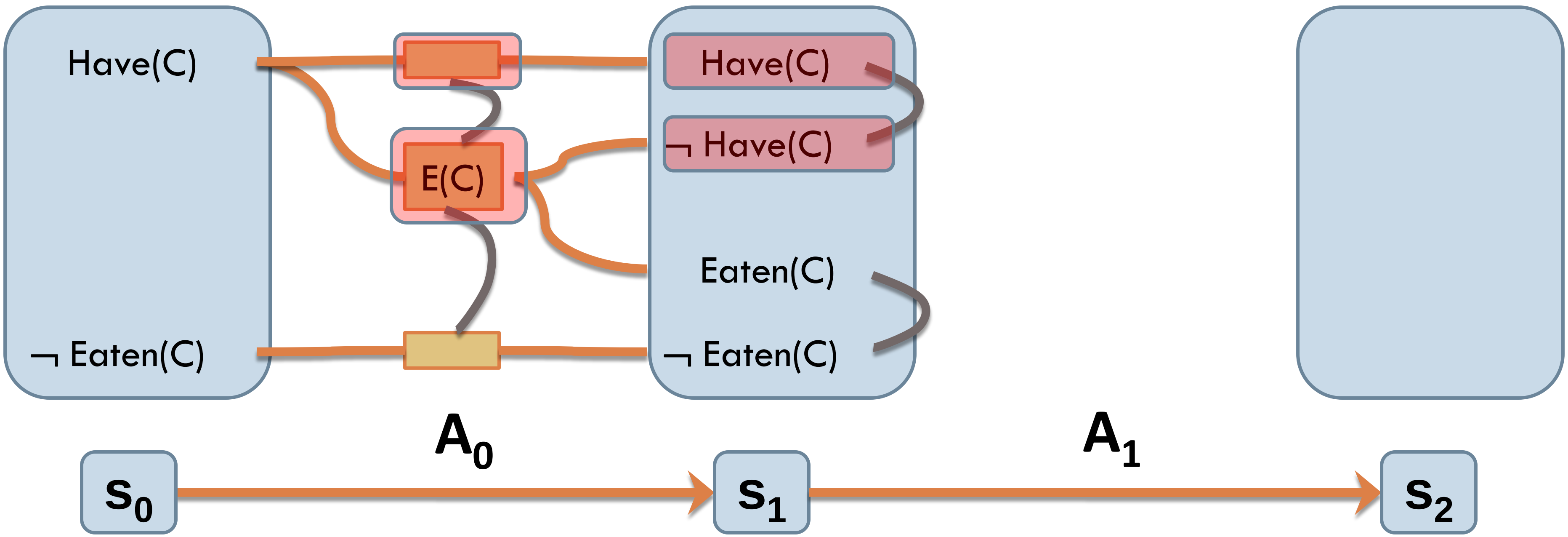
Planning graphs

- Two actions have an interference when:
  - One effect of one action is the negation of a precondition for the other action
    - E.g., action E(C) and the persistent action for Have(C)



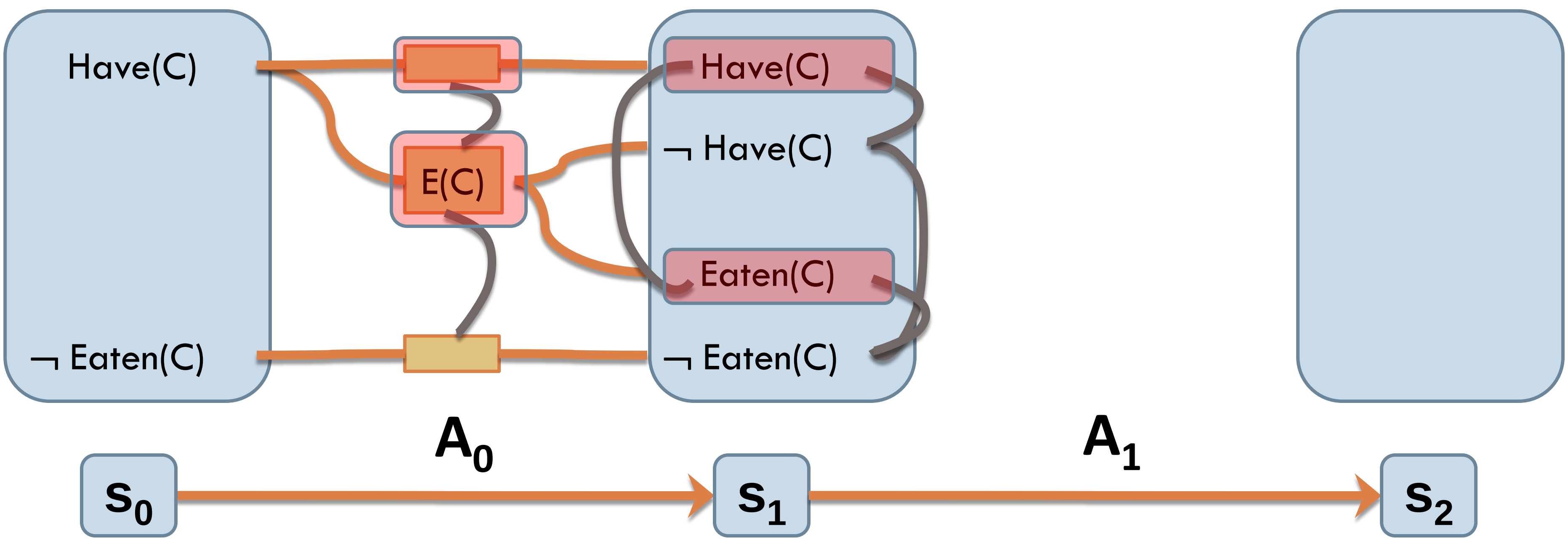
Planning graphs

- Two **literals** have inconsistent support when:
  - One literal is the negation of the other literal
    - E.g., ¬Have(C) and Have(C)



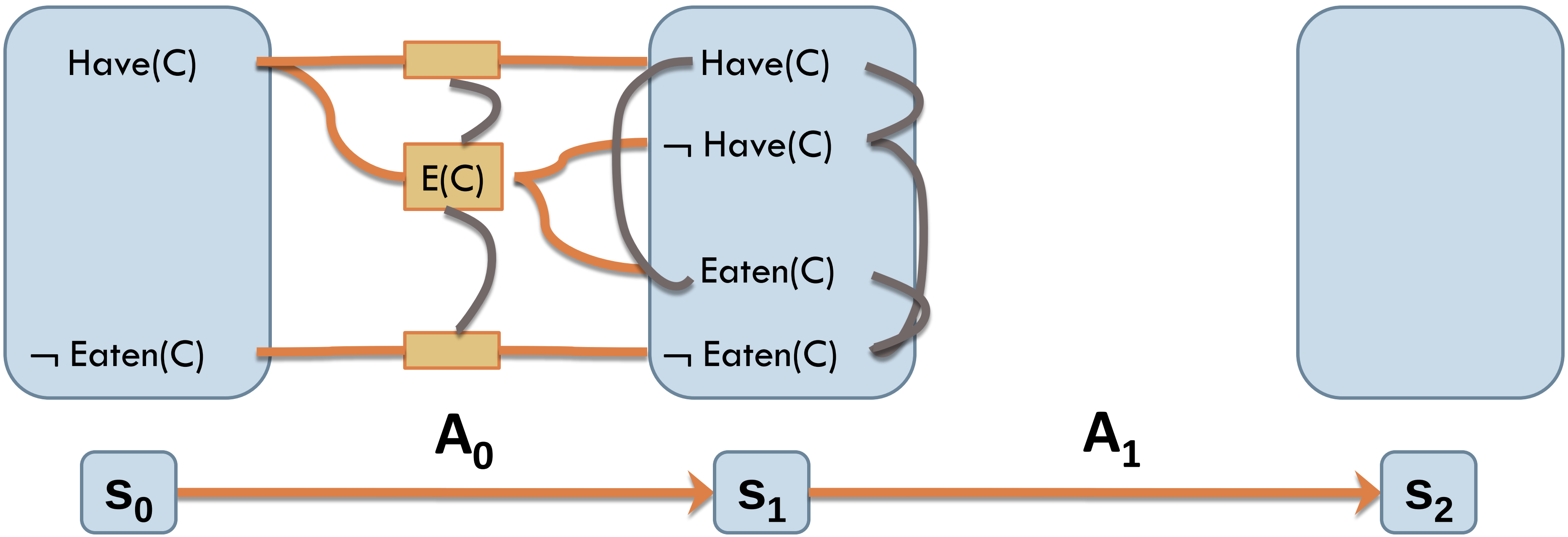
Planning graphs

- Two **literals** have inconsistent support when:
  - Every possible pair of action that have these literals as effects are marked as mutually exclusive



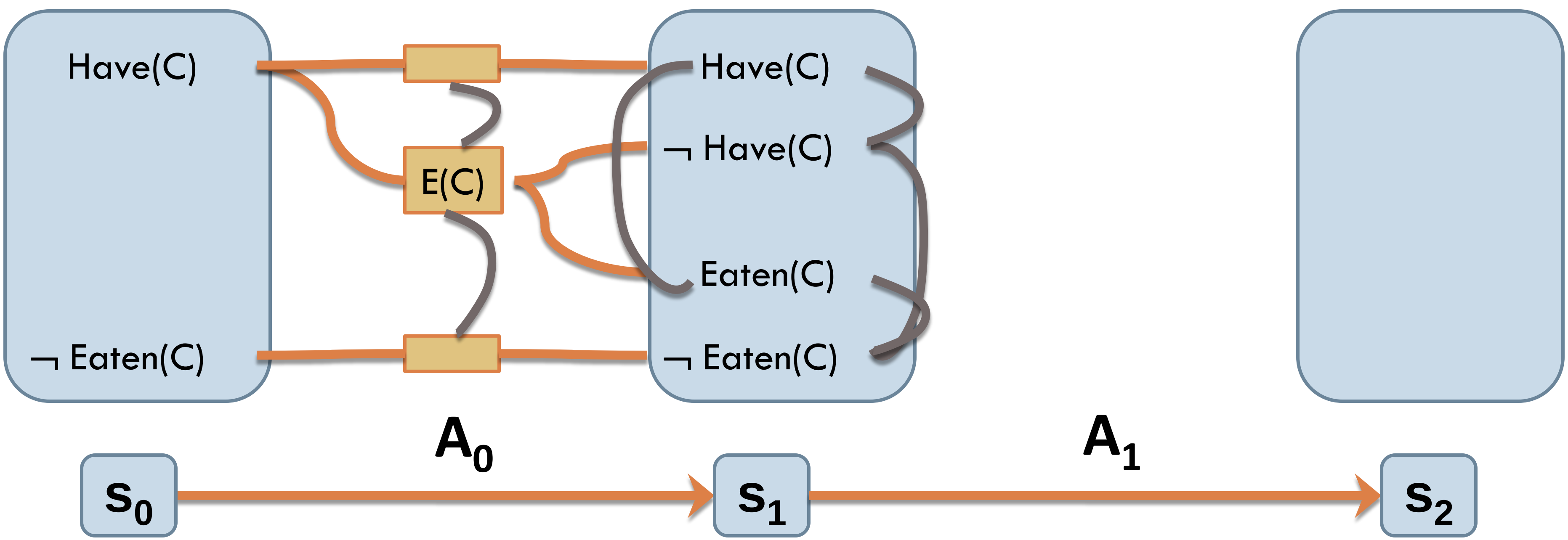
Planning graphs

- Two actions have competing needs when:
  - A precondition of one action is mutually exclusive with a precondition of the other action
    - Does not arise in this domain



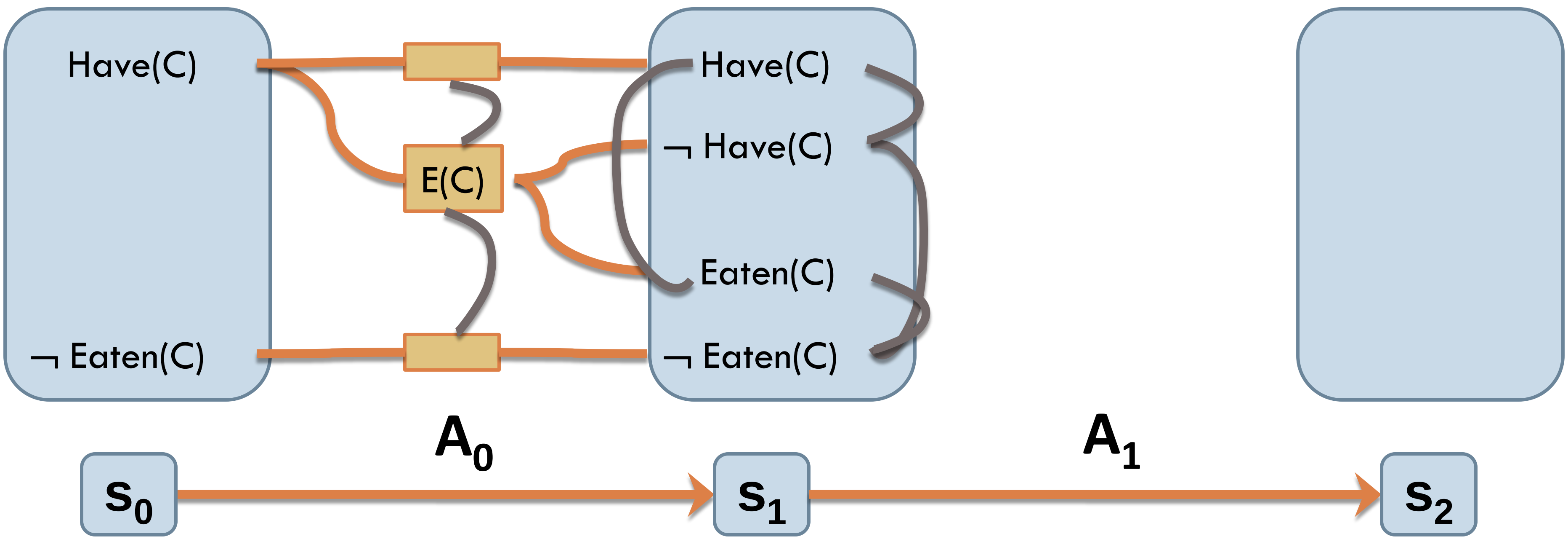
Planning graphs

- Level 0
  - We are (finally) done!



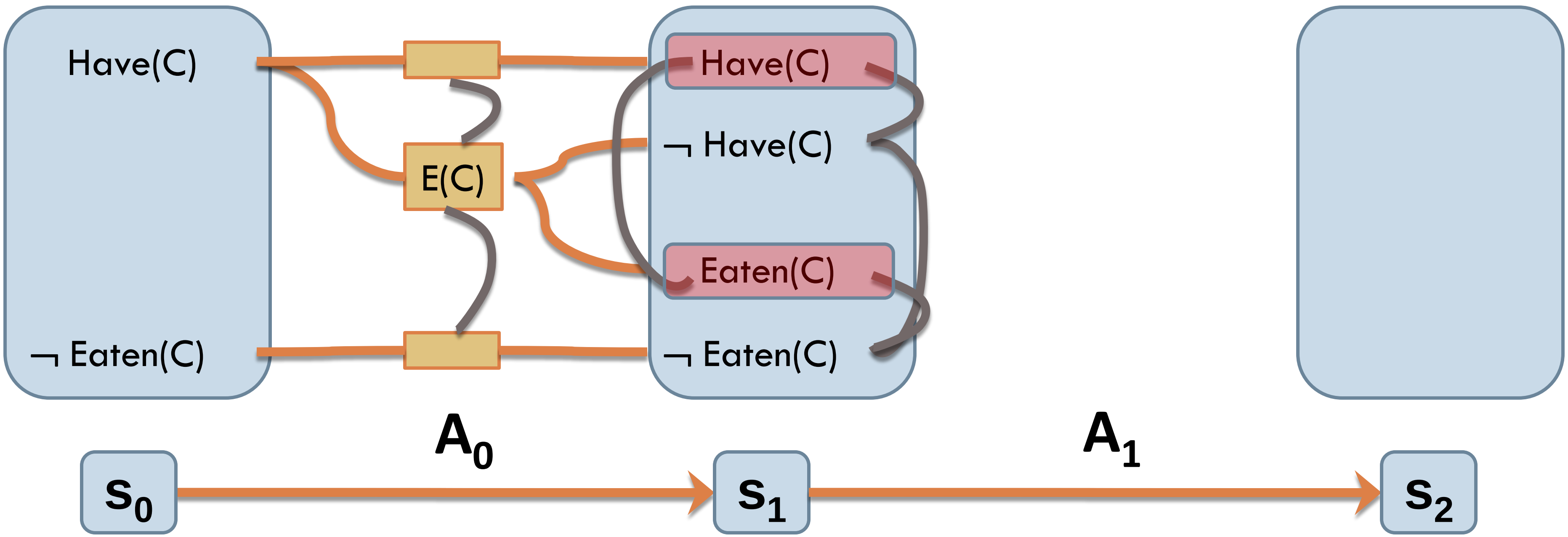
Planning graphs

- What kind of information does the graph provide so far?



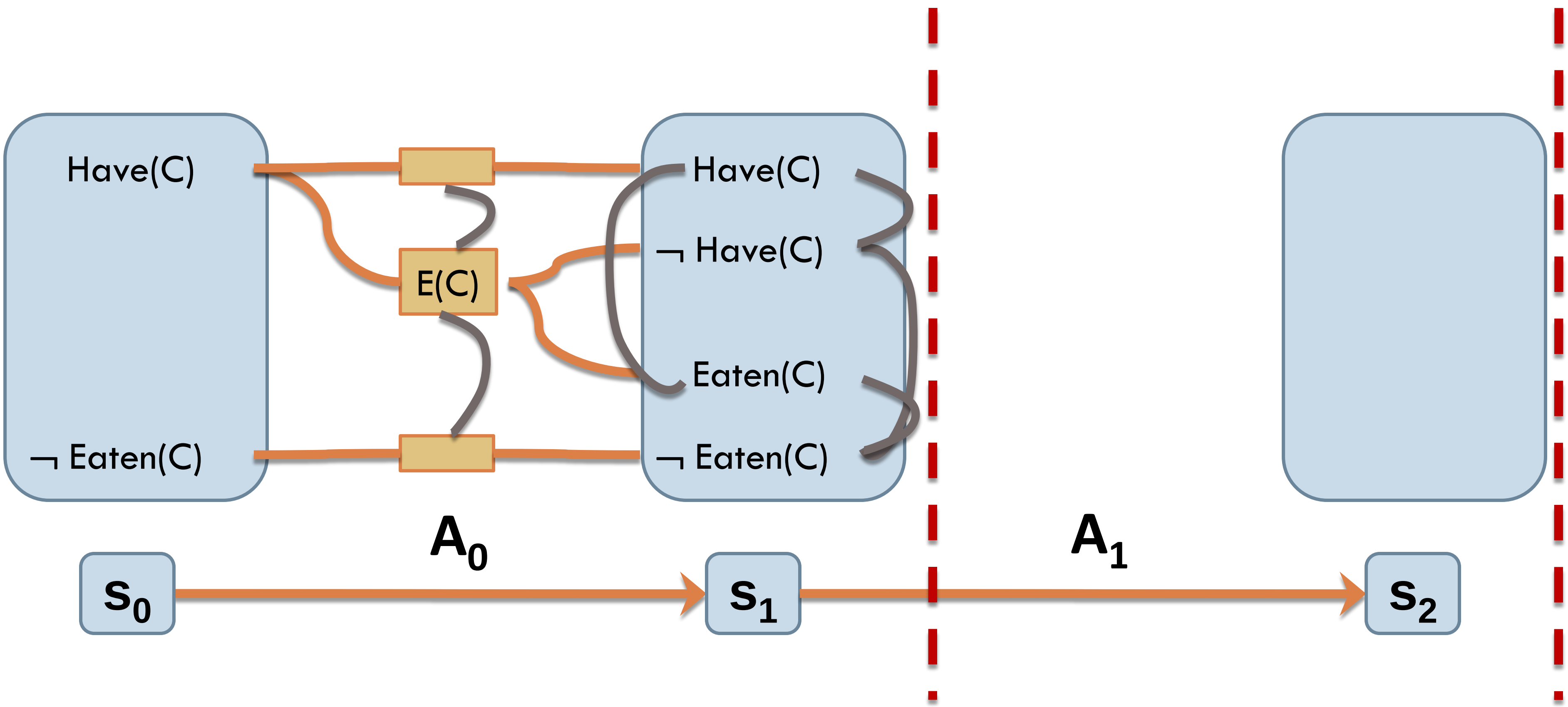
Planning graphs

- A pair of mutually exclusive literals cannot be realized from the actions of Level 0!
  - E.g., the goal cannot be achieved with these actions



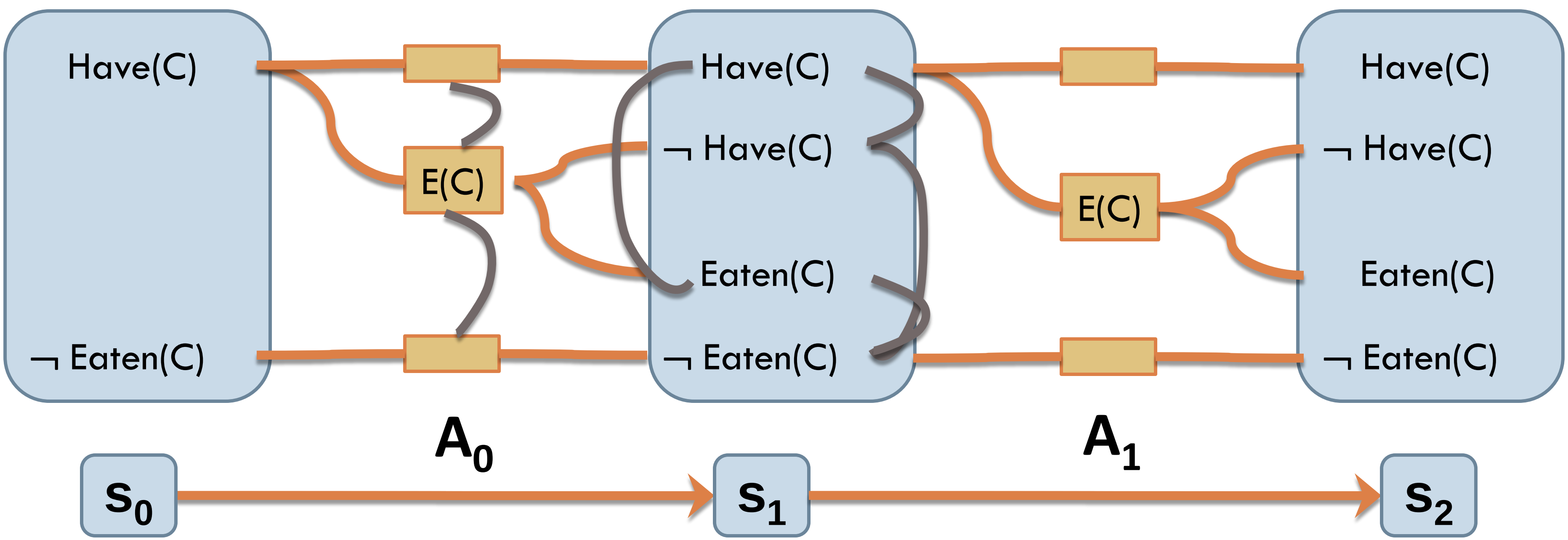
Planning graphs

- Level 1
  - We will specify $\mathbf{A_1}$ and $\mathbf{S_2}$



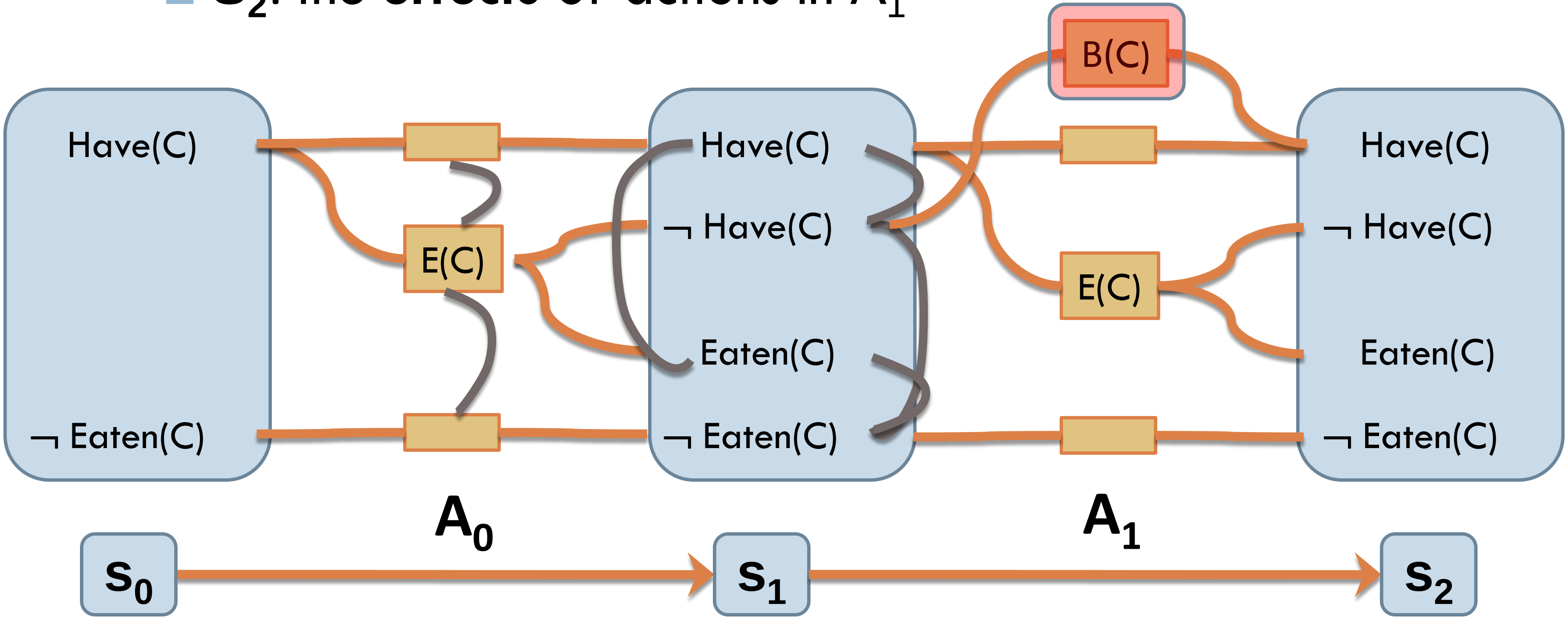
Planning graphs

- Level 1
  - $\mathbf{A_1}$: the **applicable actions** in S_1 (at least those in $\mathbf{A_0}$)
  - $\mathbf{S_2}$: the **effects** of actions in A_1 (at least those in $\mathbf{S_1}$)



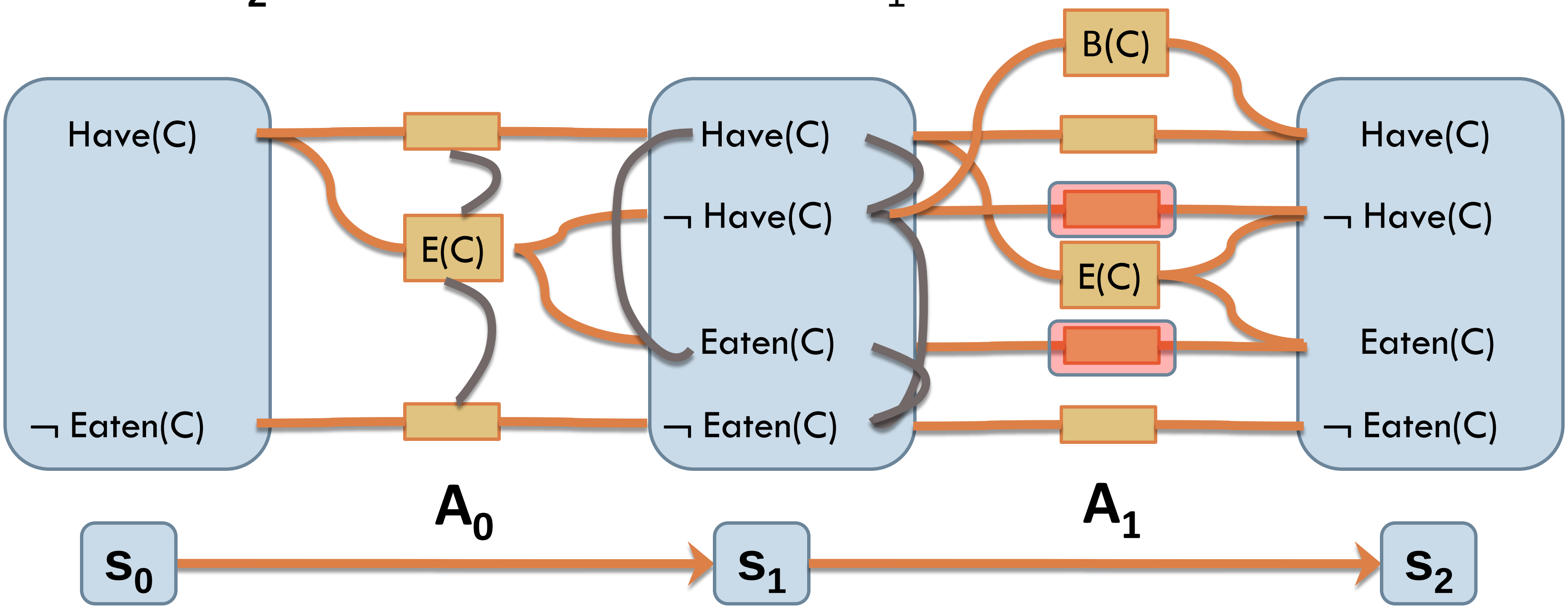
Planning graphs

- Level 1
  - A_1: the **applicable actions** in S_1 (and more!)
  - S_2: the **effects** of actions in A_1



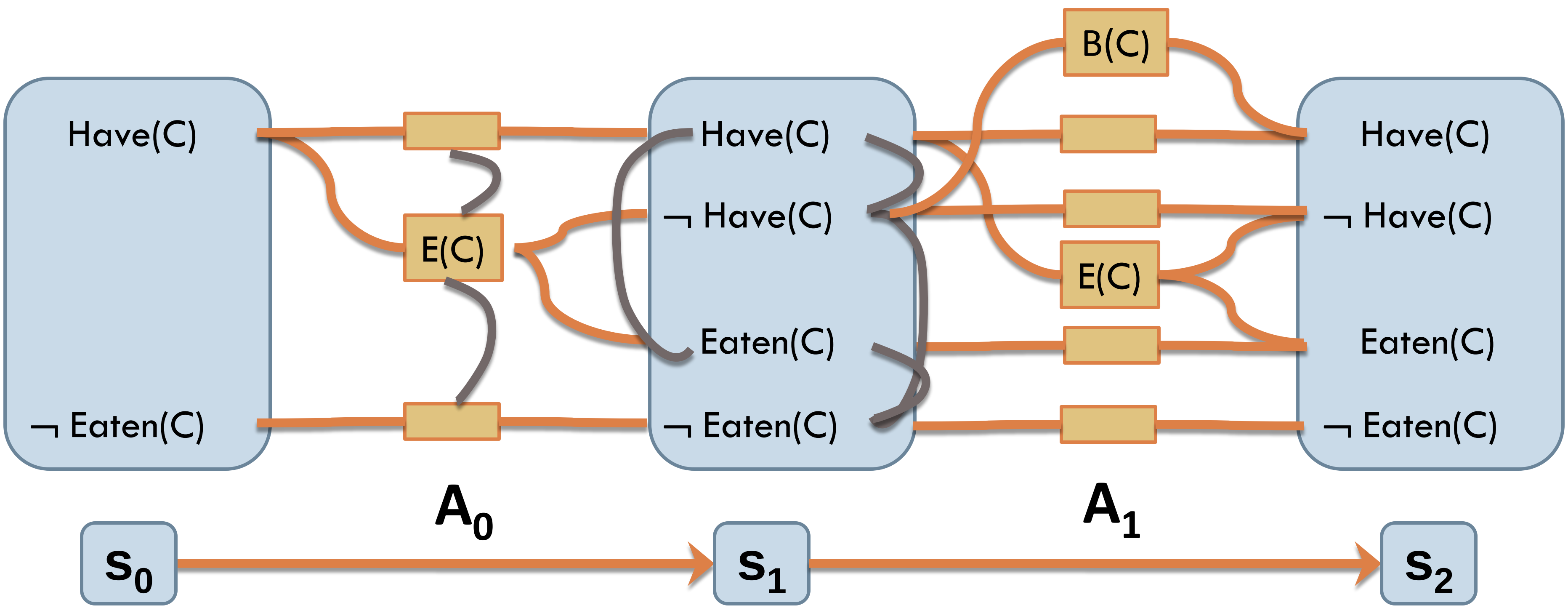
Planning graphs

- Level 1
  - A_1: the **applicable actions** in S_1 (and more!)
  - S_2: the **effects** of actions in A_1



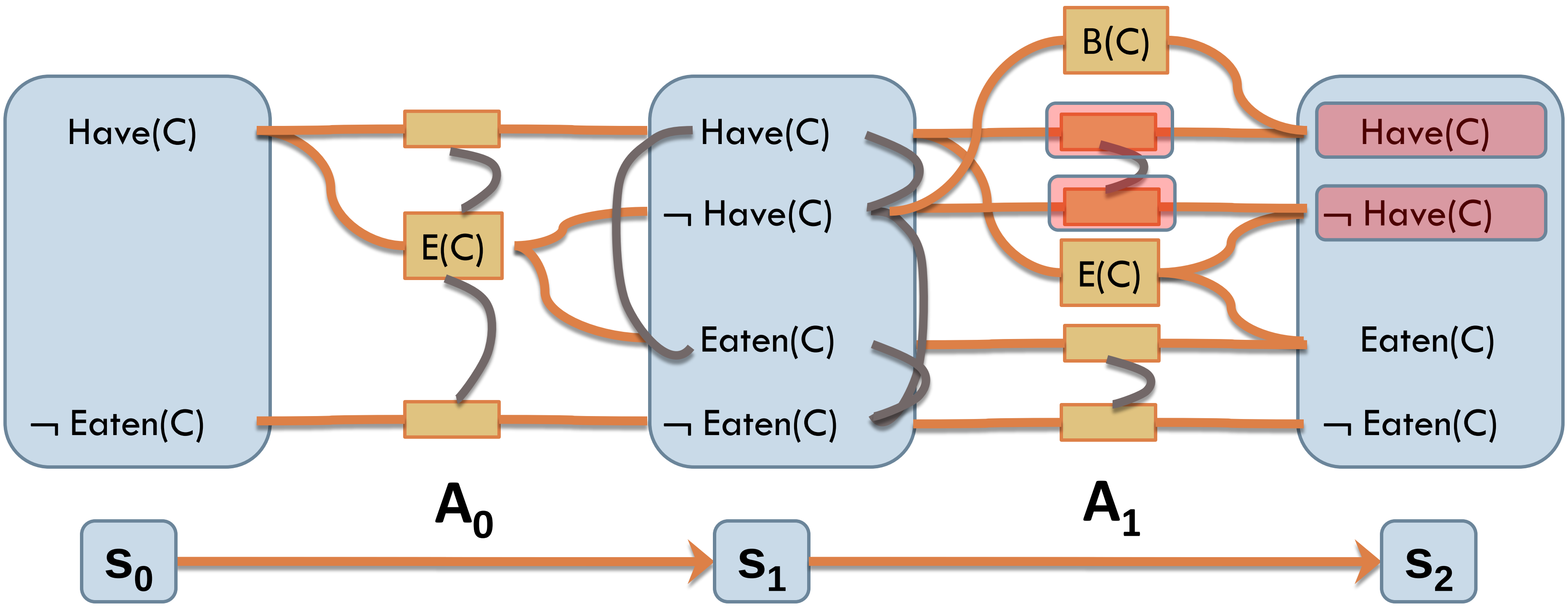
Planning graphs

- Level 1
  - Mutual exclusive links



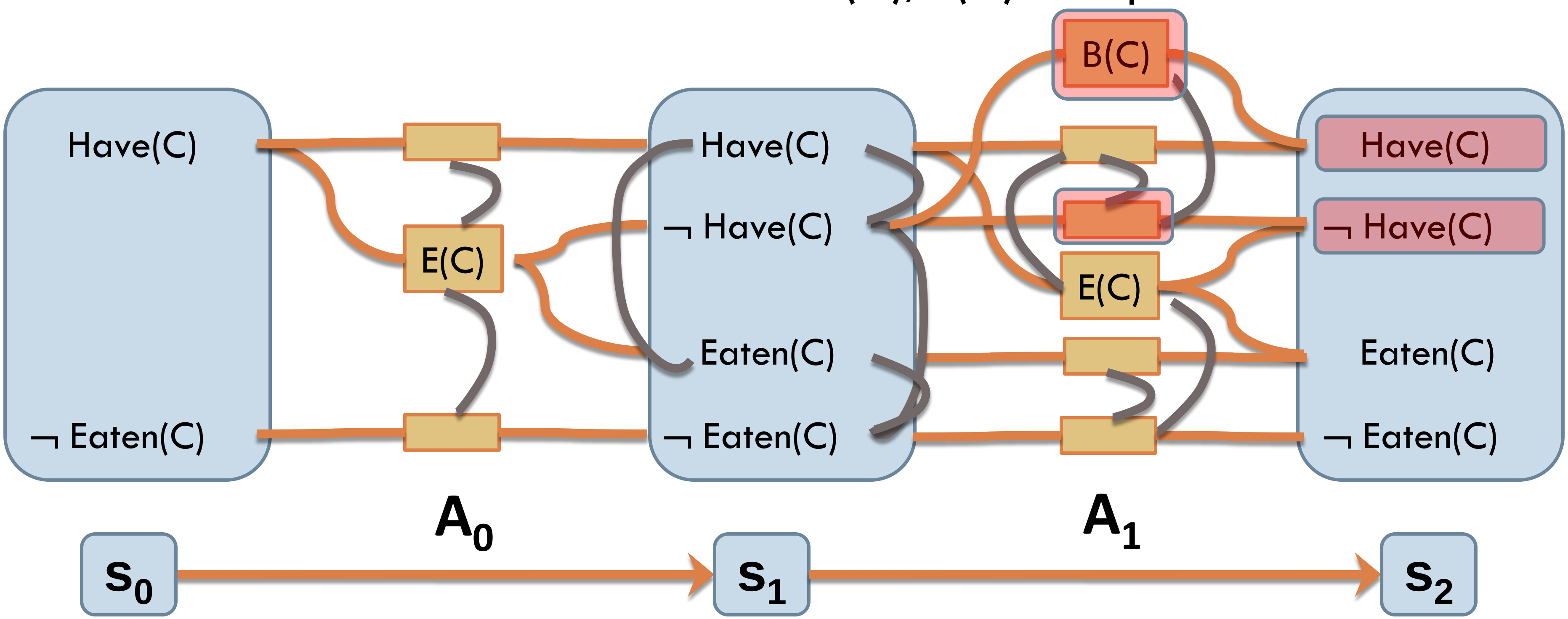
Planning graphs

- Level 1
  - Mutual exclusive links
  - Inconsistent effects between persistence actions



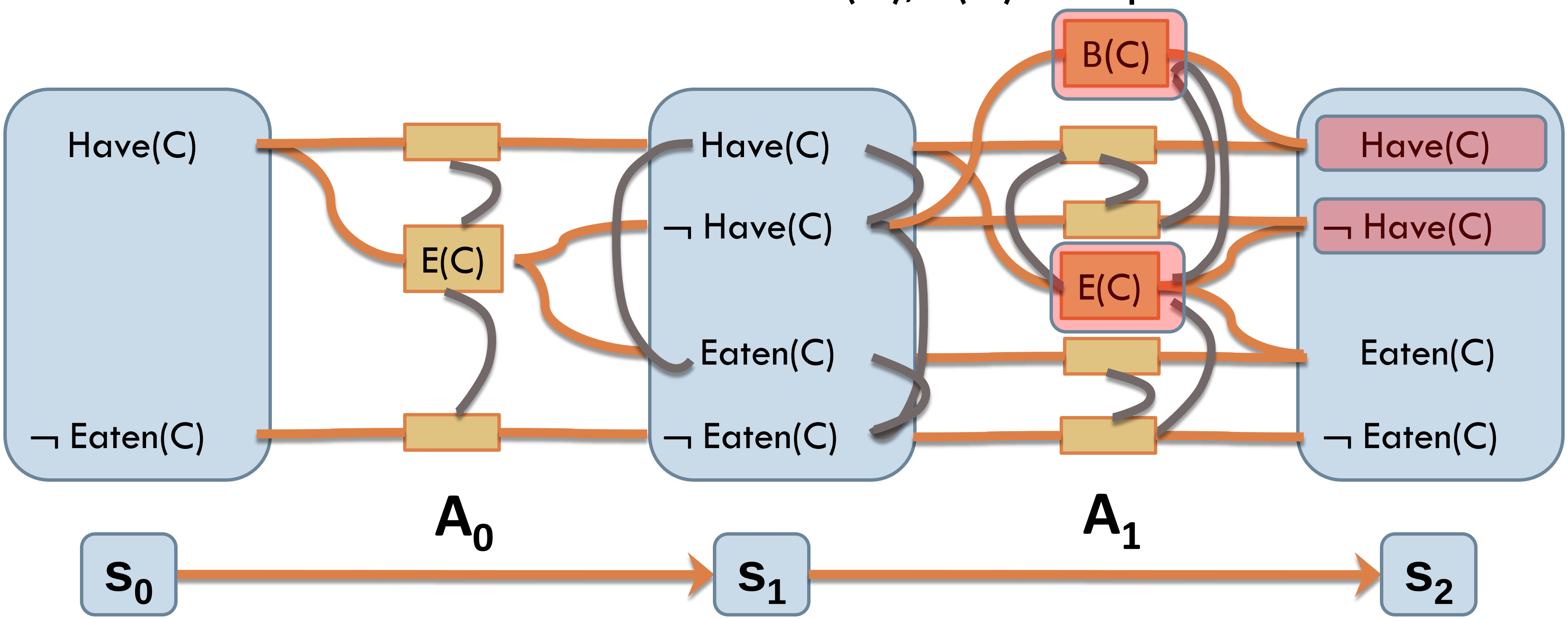
Planning graphs

- Level 1
  - Mutual exclusive links
  - Inconsistent effects between B(C), E(C) and persistence actions



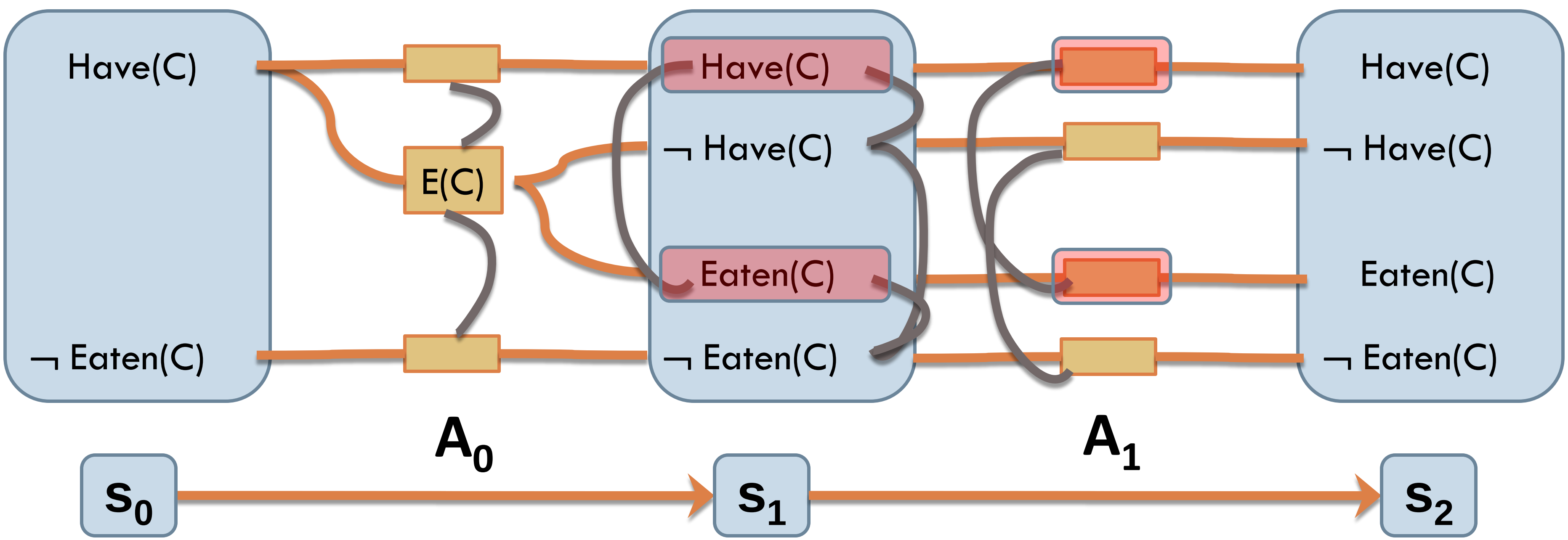
Planning graphs

- Level 1
  - Mutual exclusive links
  - Inconsistent effects between B(C), E(C) and persistence actions



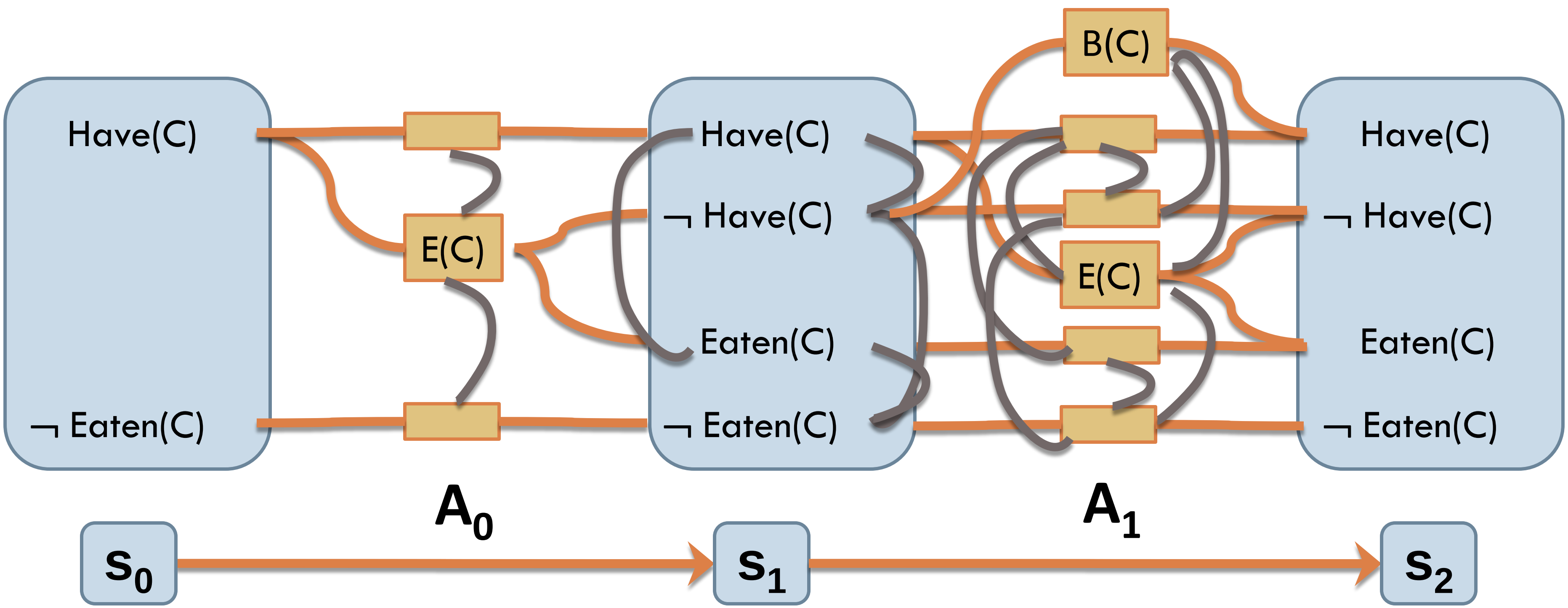
Planning graphs

- Level 1
  - Mutual exclusive links
  - Competing needs between persistence actions!



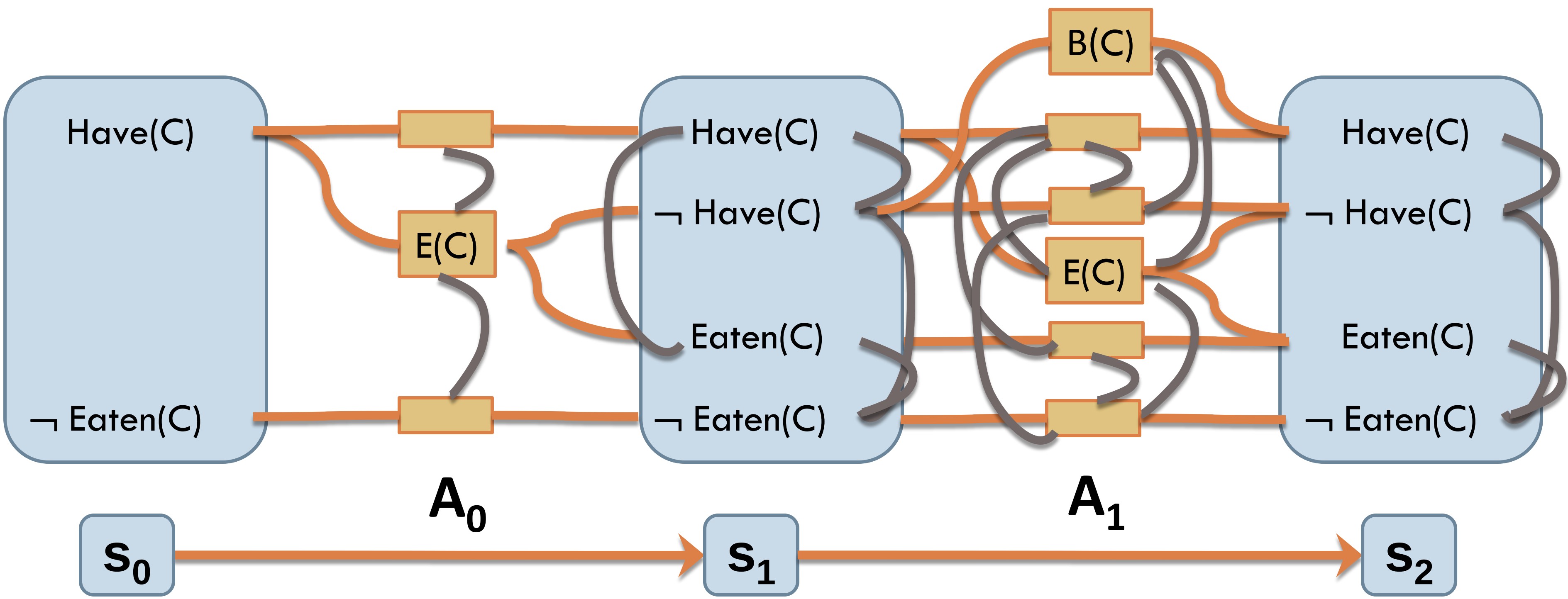
Planning graphs

- Level 1
  - Mutual exclusive links
  - No more mutexes between actions



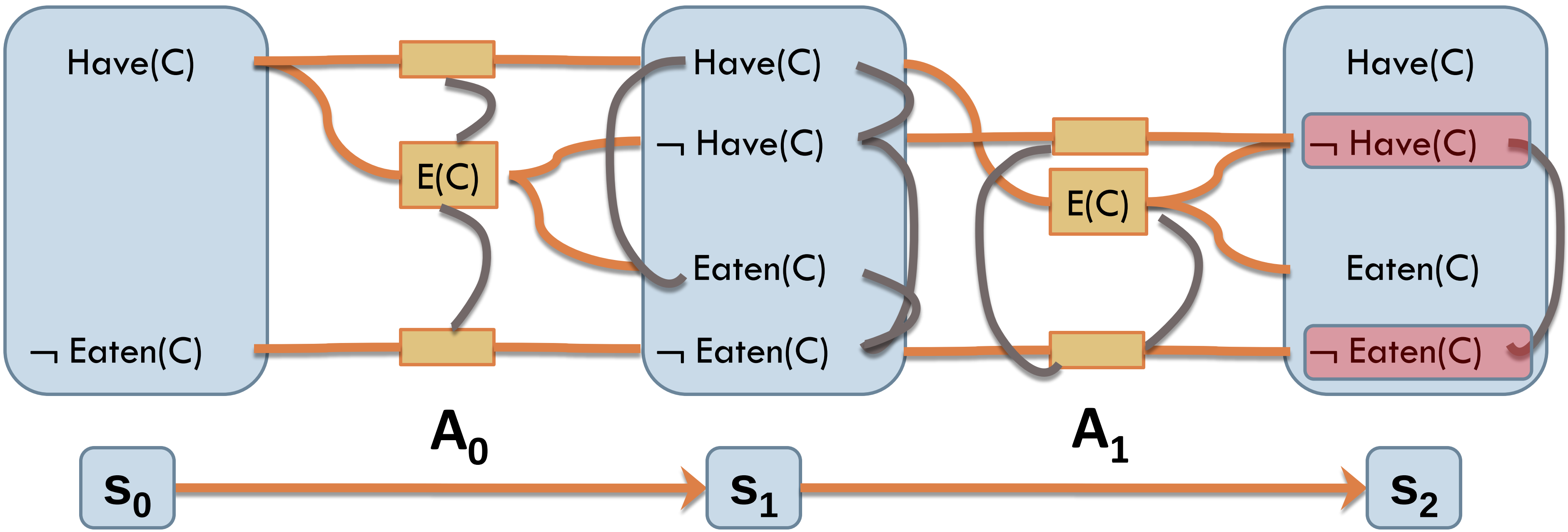
Planning graphs

- Level 1
  - Mutual exclusive links
  - There are mutexes between literals in S_2 though



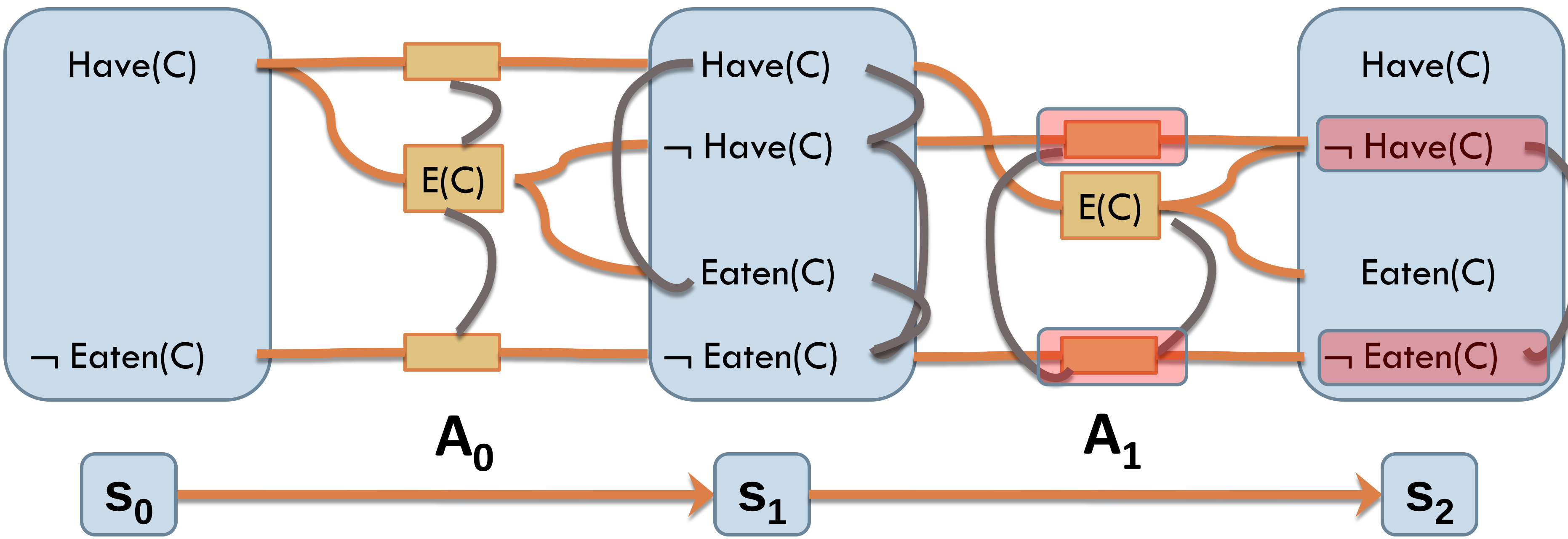
Planning graphs

- Level 1
  - Mutual exclusive links
  - Between literals ¬Have(C) and ¬Eaten(C) in S_2



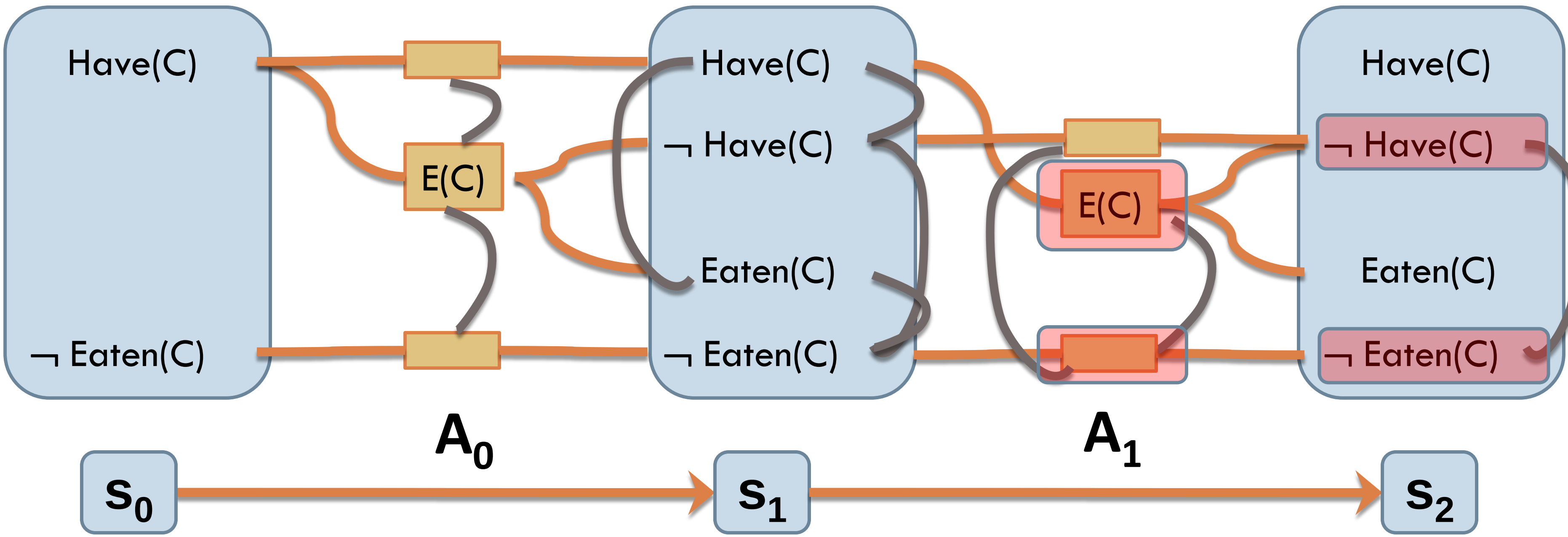
Planning graphs

- Level 1
  - Mutual exclusive links
  - Between literals ¬Have(C) and ¬Eaten(C) in S_2



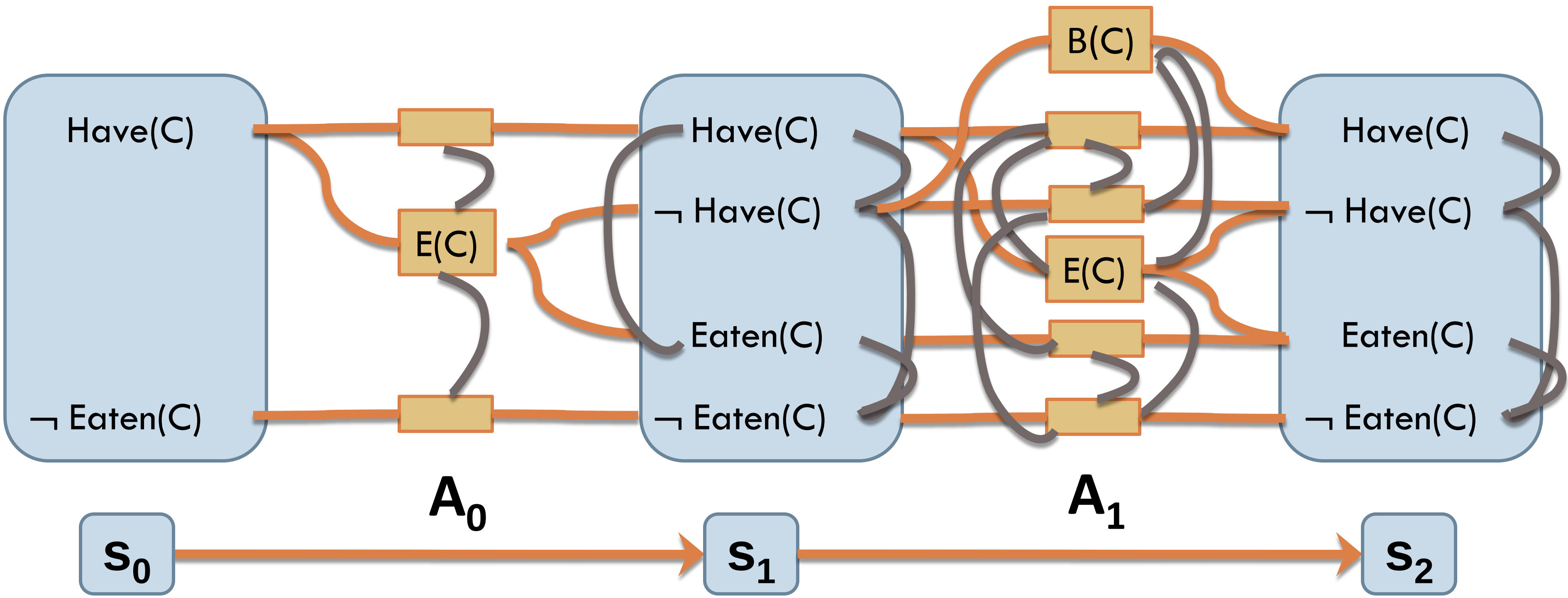
Planning graphs

- Level 1
  - Mutual exclusive links
  - Between literals ¬Have(C) and ¬Eaten(C) in S_2



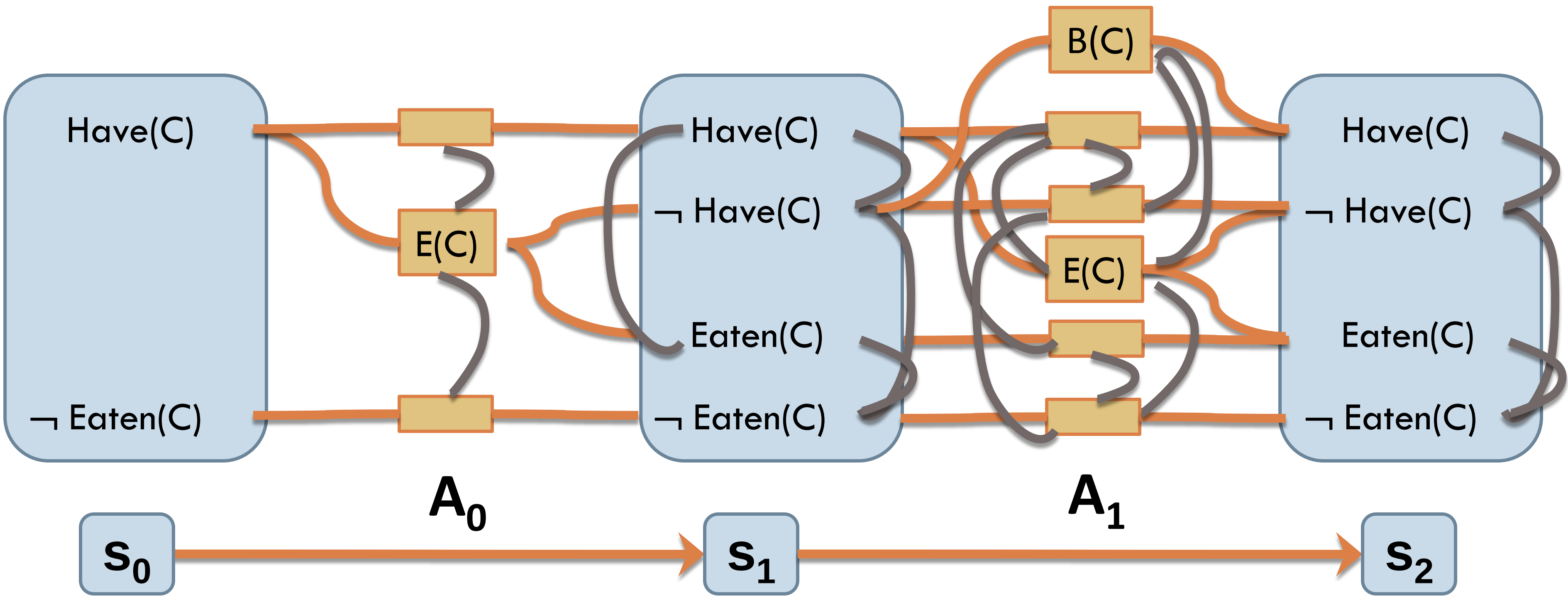
Planning graphs

- Level 1
  - We are (finally) done!



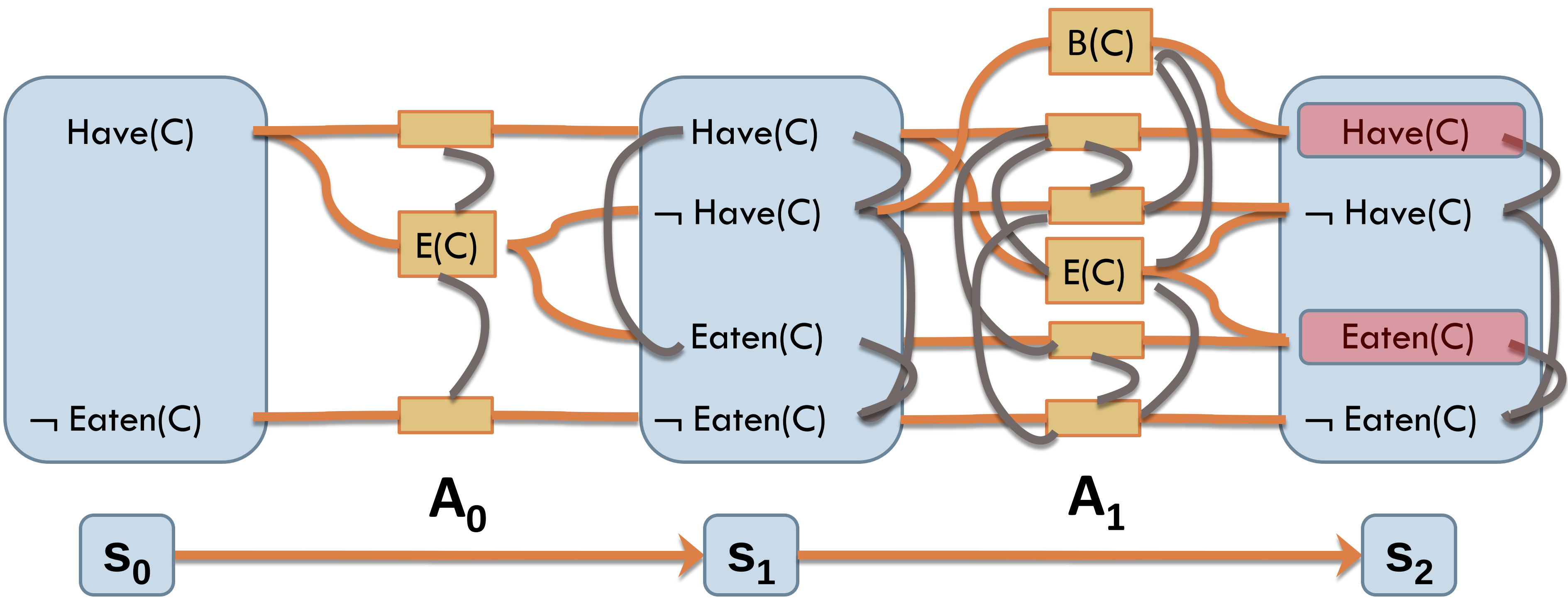
Planning graphs

- What information can we get from the graph now?



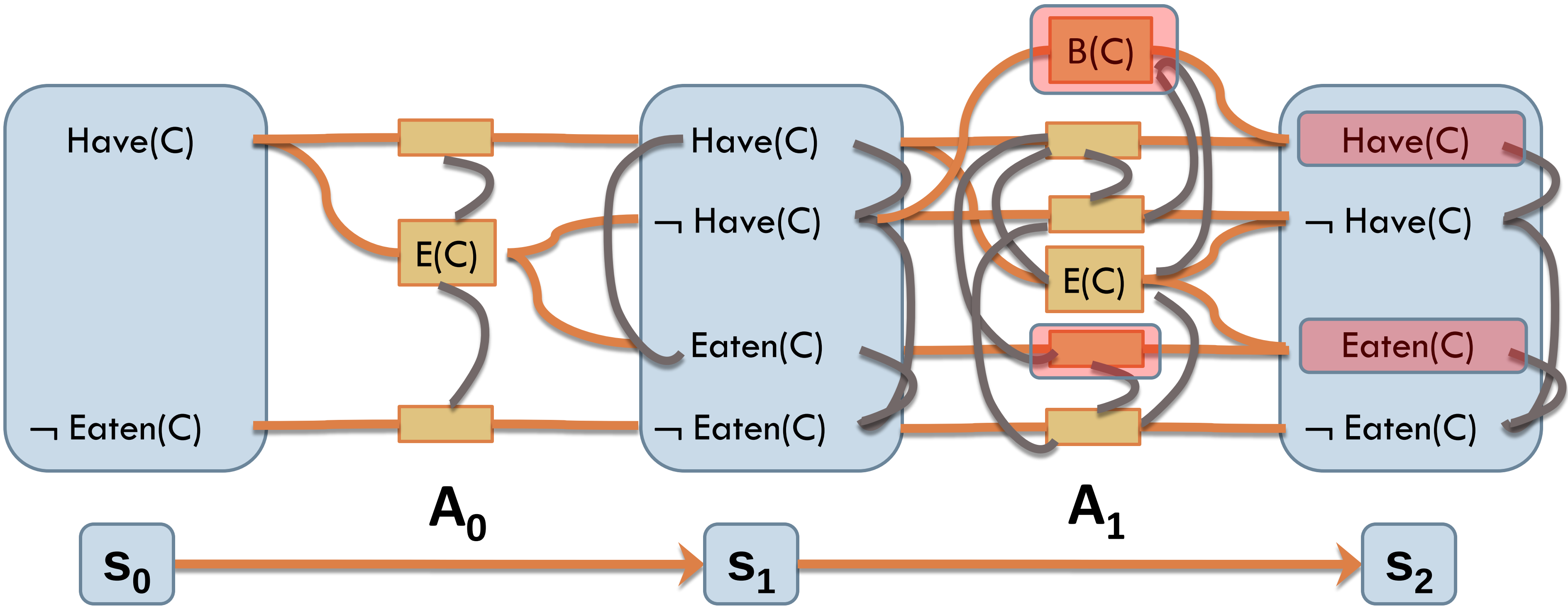
Planning graphs

- What information can we get from the graph now?
  - Note that literals Have(C) and Eaten(C) **are not mutually exclusive** in S_2 !!!



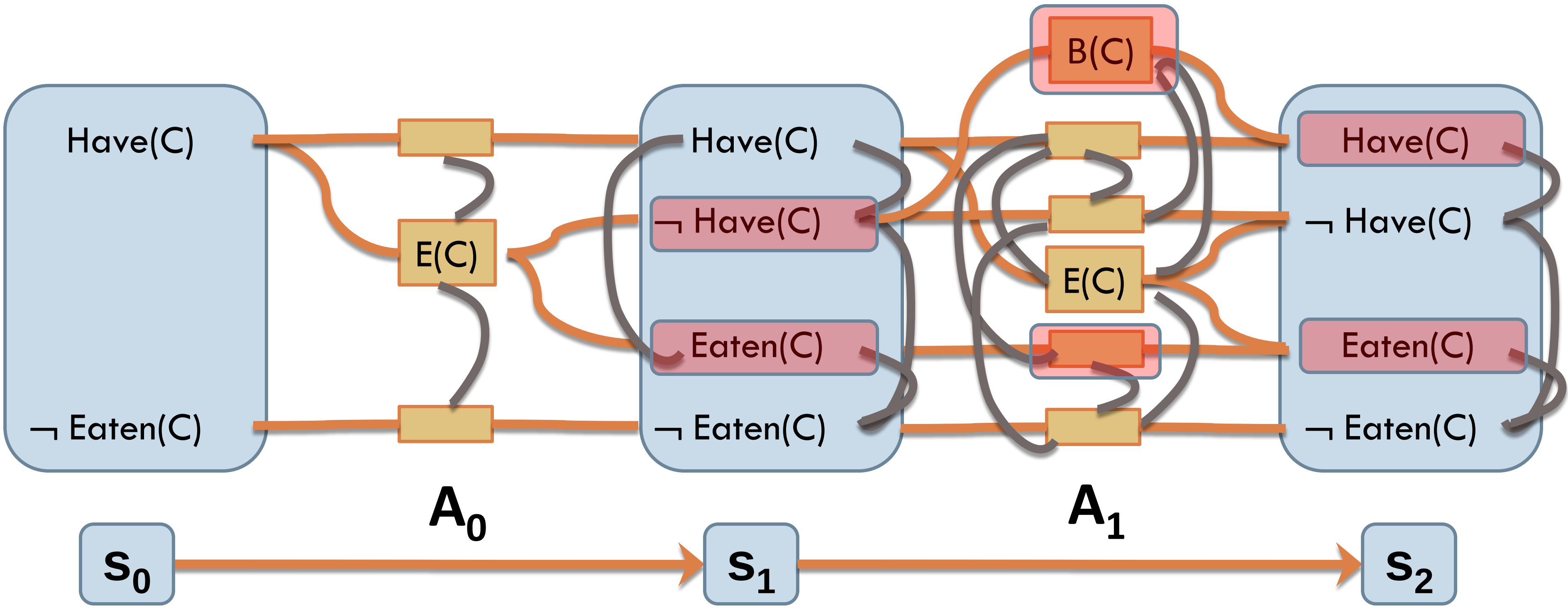
Planning graphs

- What information can we get from the graph now?
  - Note that literals Have(C) and Eaten(C) **can be realized** in A_1 by the actions {B(C), persistence of Eaten(C)}



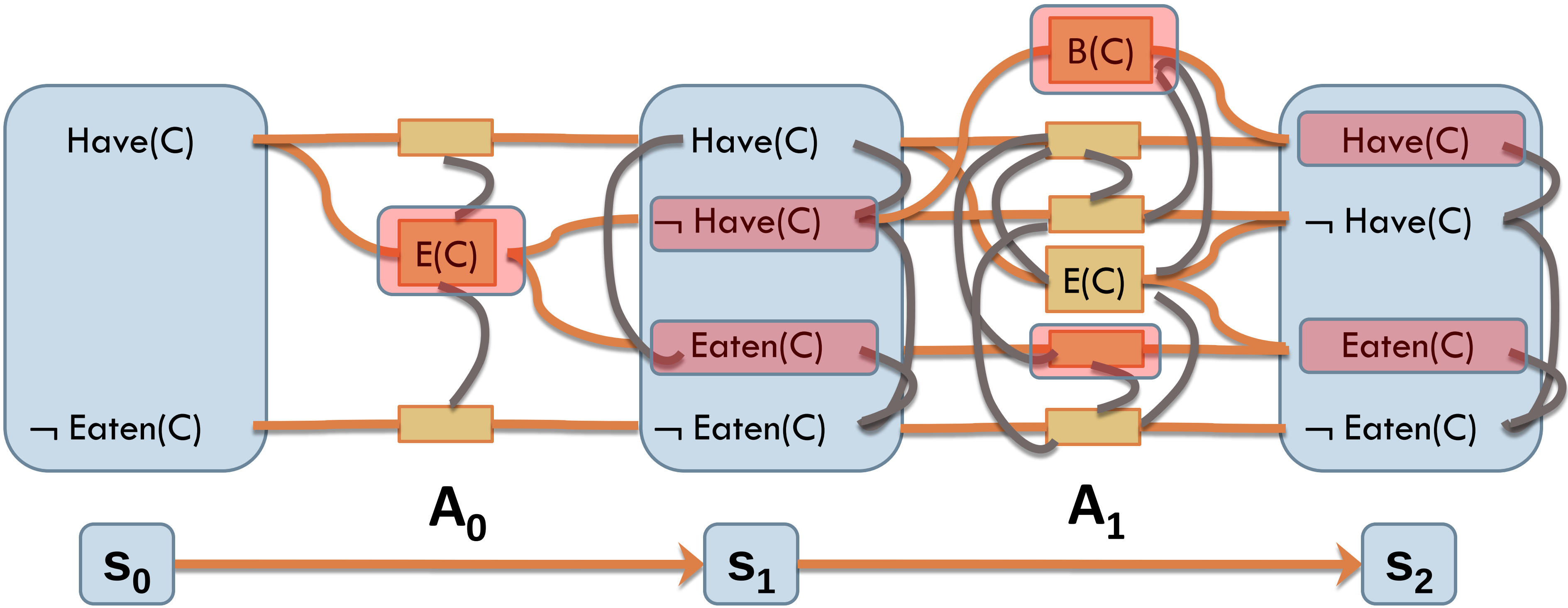
Planning graphs

- What information can we get from the graph now?
  - In turn actions {B(C), persistence of Eaten(C)} **require** that ¬ Have(C) and Eaten(C) hold in S_1



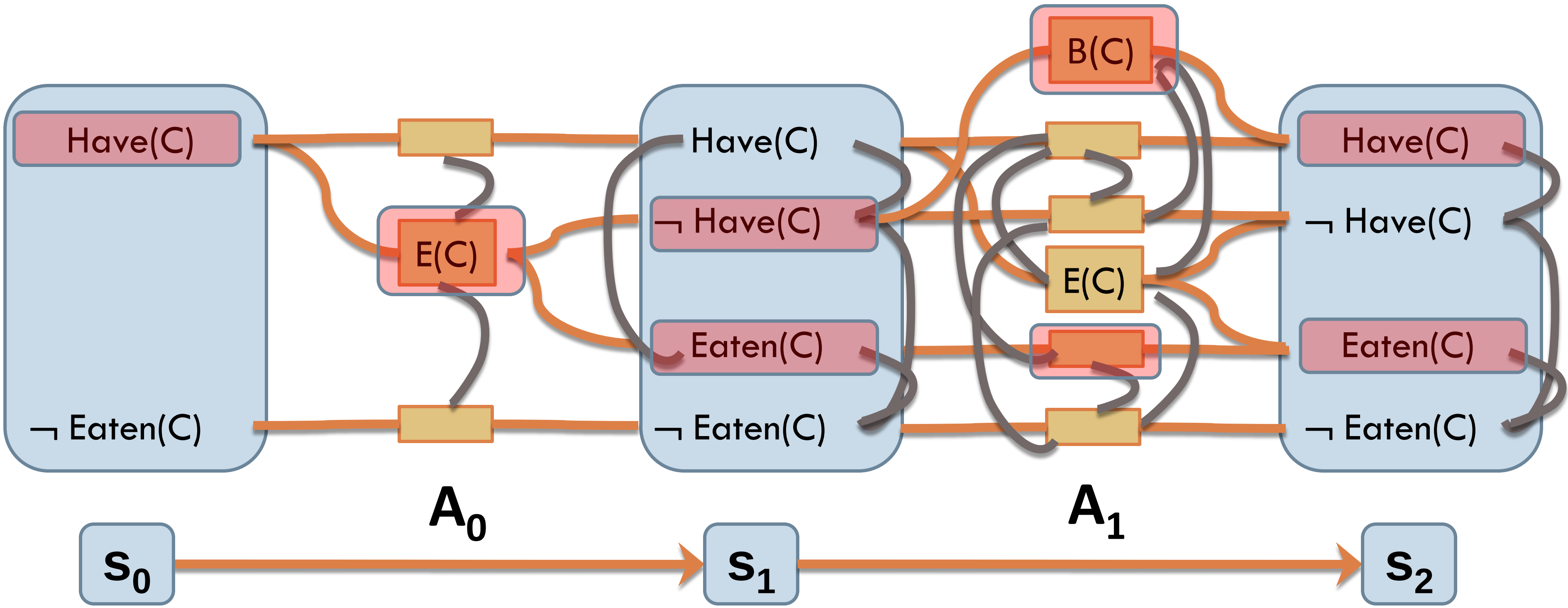
Planning graphs

- What information can we get from the graph now?
  - Note that literals Have(C) and Eaten(C) **can be realized** in A_0 by the action E(C)



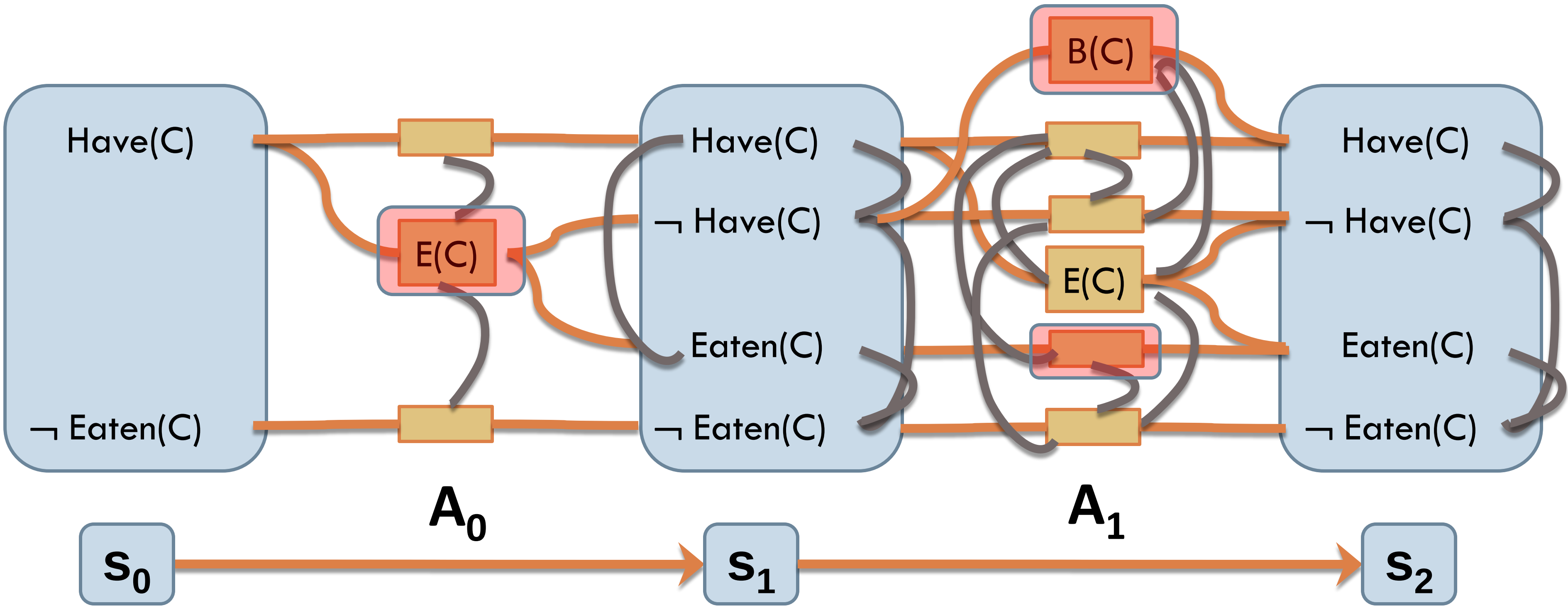
Planning graphs

- What information can we get from the graph now?
  - In turn E(C) requires that Have(C) holds in S_0 which is true!



Planning graphs

- What information can we get from the graph now?
  - So, actions **{E(C)}** and **{B(C), persistence of Eaten(C)}** can actually **achieve the goal!**



Planning graphs

- Planning graph

Planning graphs

- Planning graph
  - **When do we stop calculating levels?**
  - When two consecutive levels are identical *
  - **How do we know this will happen at some point?**
  - Literals and actions increase monotonically, while mutexes decrease monotonically (why is this so?)

Planning graphs

- Planning graph
  - Special **data structure**
  - Easy to compute: **polynomial complexity!**
  - Can be used by the **GRAPHPLAN** algorithm to **search for a solution** (following similar reasoning as in the example)
  - Can be used as a **guideline for heuristic functions** for progressive planning that are more accurate than the ones we sketched in Lecture 2

Bibliography

- Material
  - Artificial Intelligence: A Modern Approach 2nd Ed. Stuart Russell, Peter Norvig. Prentice Hall, 2003 Section 11.4