# Preface

Artificial Intelligence (A.I.) is a term which has been coined in the late 50s by a group of young and enthusiastic researchers. Since then, many other researchers kept the flame intact and have added their name to the hall of fame.

A.I. Planning  has been proposed in 1971 by Richard Fikes and Nils Nilsson. After decades of exploratory systems, the first conference on this scientific domain emerged in the USA in 1992: AIPS for A.I. Planning Systems. A little bit later emerged the EWPS forum, the European Workshop on Planning Systems, which became the European Conference on Planning (ECP). In the early 2000, both conferences merged under the term ICAPS, for International Conference on Automated Planning and Scheduling, which takes place alternately on both sides of the Atlantic ocean.

Constraint Programming, another area of A.I., has been proposed in 1976 by Jean-Louis Laurière in his *Doctorat d'Etat* (now HdR). Although its exact birth date is debated, his ideas led to forums now known as CP, for Constraint Programming and to the Constraints journal.

I hope that the reader will find in the following pages a smooth and soft introduction to these two domains, which are taught in 2019 at B.S. level in many universities worldwide.

# Decision in Artificial Intelligence

An introduction to A.I. Planning and
to Constraint Programming

# Part I : A.I. planning

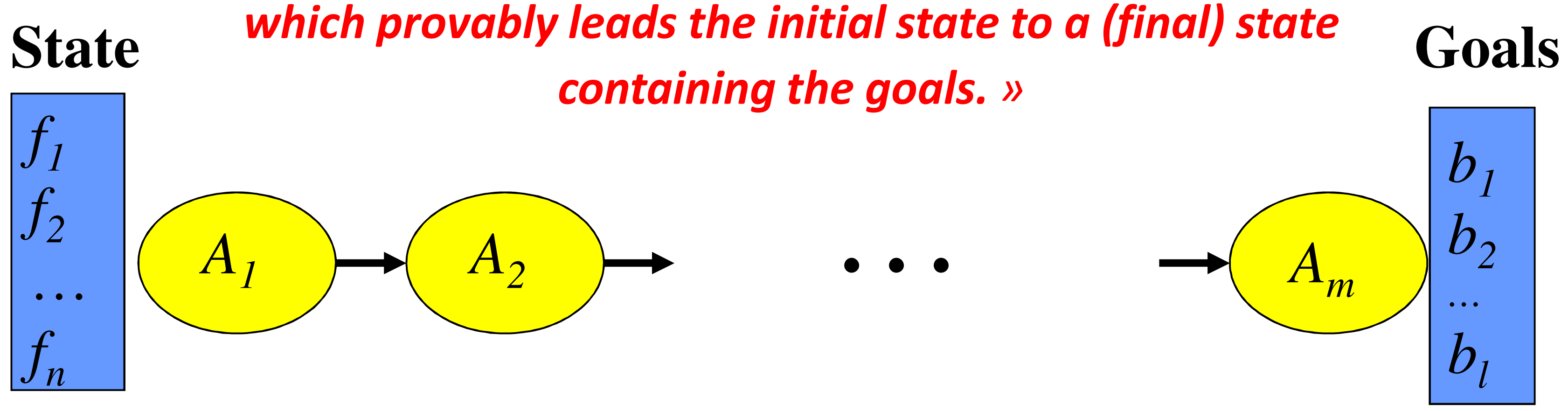
Philippe Morignot

pmorignot@yahoo.fr

## Table of contents

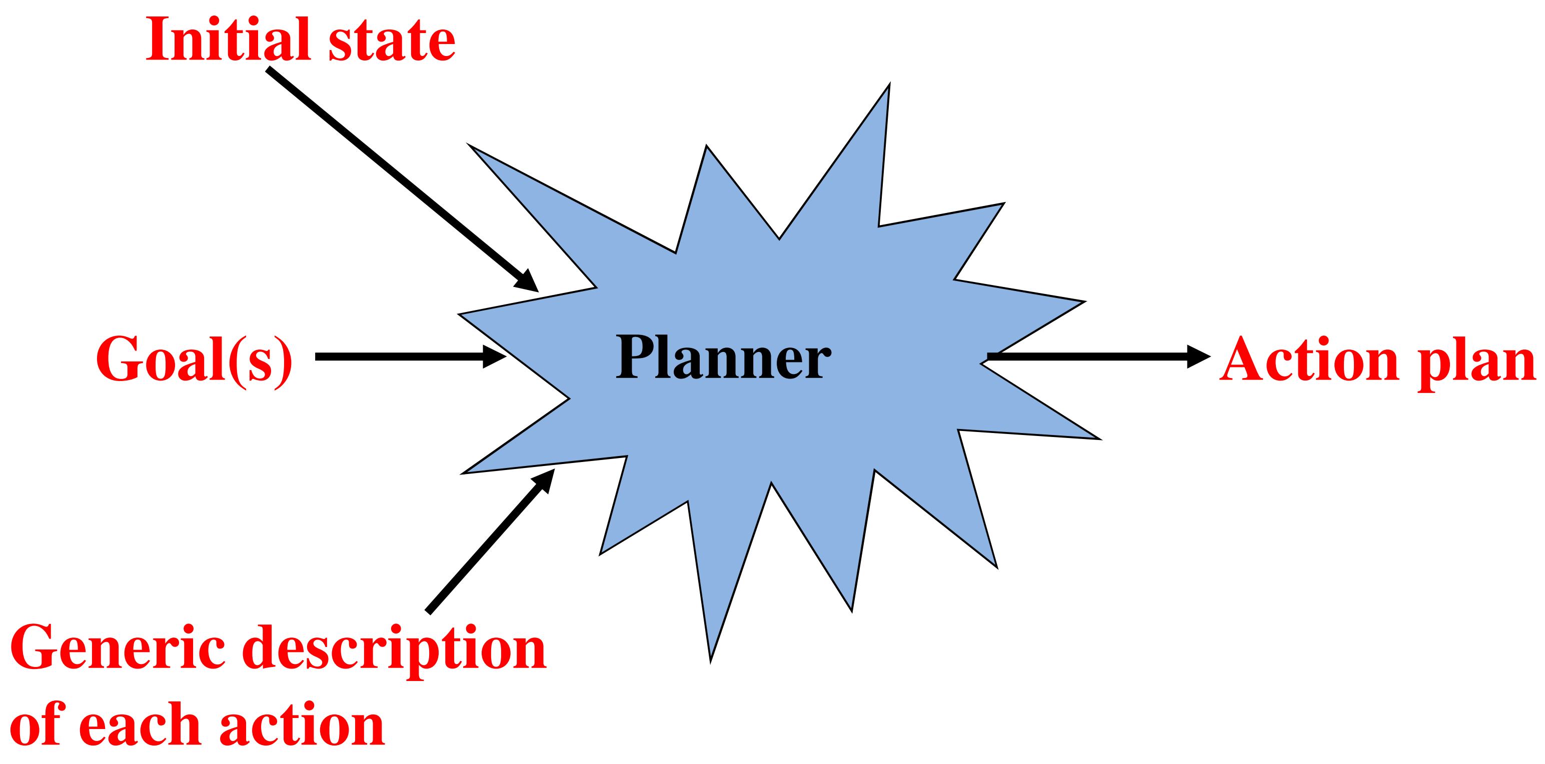
# Statement of the problem

***« Given possible generic actions,***
***an initial state and goals,***
***find a sequence of instantiated actions,***
***which provably leads the initial state to a (final) state***
***containing the goals. »***



- « ***Action planning*** » / « ***plan synthesis*** » / « ***generation of action plan*** » : activity of constructing a plan.
- « ***Planner*** » / « ***task planner*** » / « ***action planner*** » : computer program which solves this problem.
  - Different from « path planner » in Robotics.

# An action planner



# Difficulty

- **Crane domain:**
  - 1 crane, *a* locations, *b* trucks, *c* container stacks, *d* containers.

- If $a = 5$, $b = 3$, $c = 3$, $d = 100$, then ~ $10^{277}$ states.
- Classical planning is NP.
- **All the states cannot be enumerated.**

## Assumptions

- A1 : ***the agent is the sole cause of change in the environment***.
  - No other agent, artificial or human.
- A2 : ***the environment is totaly observable, the agent perfectly knows it***.
  - The agent does not reason (e.g., plan) on things it does not know.
- A3 : ***the environment is static.***
  - Even if the environment can have behavior laws, it does not spontaneously move.

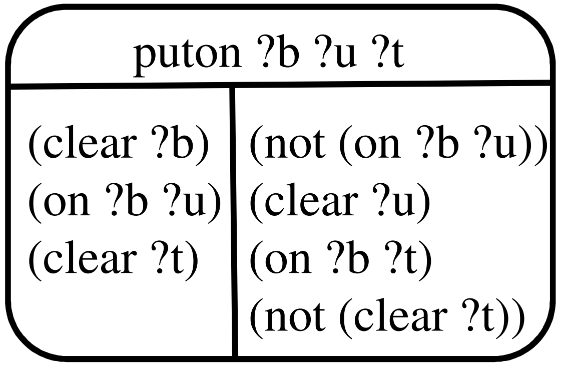
## Planning Domain Definition Language (PDDL)

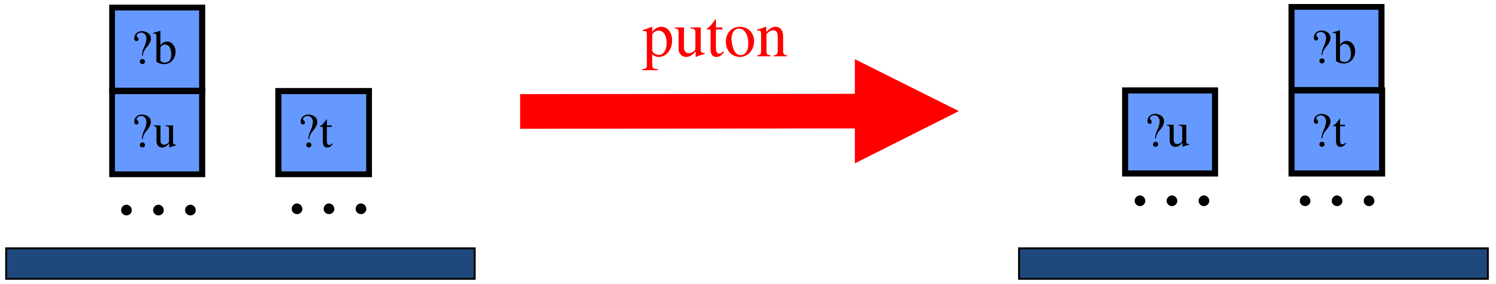
- Representation language which defines:
  - a domain: operators
  - a problem: state and goals.
- An operator is composed of:
  - **Pre-conditions:** terms which must hold for the action to execute.
  - **Effects / post-conditions:** terms which the execution of the action changes when compared to the incoming state (ADD-LIST / DELETE-LIST).
    - A post-condition can be positive or negative.
- A term can be sometimes true and sometimes false, depending on the time at which it is considered in the plan.
  - Connector « *not* ». Ex. : (**not** (ON MOUSE PAD))
  - *« Fluent » (litteral).* Ex. : (ON MOUSE PAD)

# PDDL: Example of domain The blocks world

- Operator :

```
(:action puton
 :parameters (?b ?u ?t - block)
 :precondition (and (clear ?b)
                    (on ?b ?u))
                    (clear ?t))
 :effect (and (not (on ?b ?u)) (clear ?u)
              (on ?b ?t) (not (clear ?t))))
```





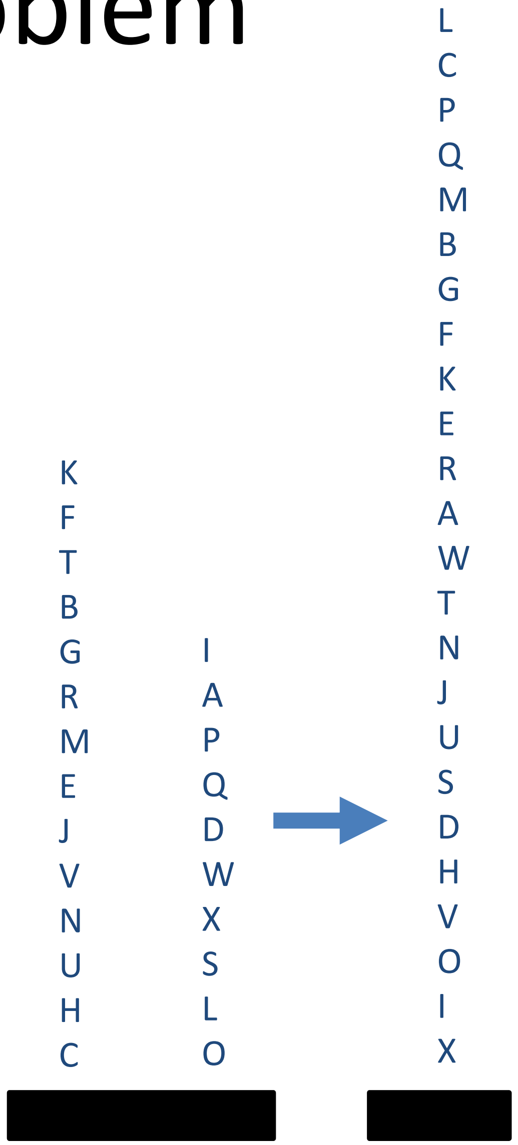
- What about the table ? And the arm ? What if several arms ? What if blocks have colors ? Or nicks ? Or multiple dimensions ? Conditionnals ? Universal quantification ?
- Qualification problem ; ramification problem.

## Planning Domain Definition Language (PDDL)

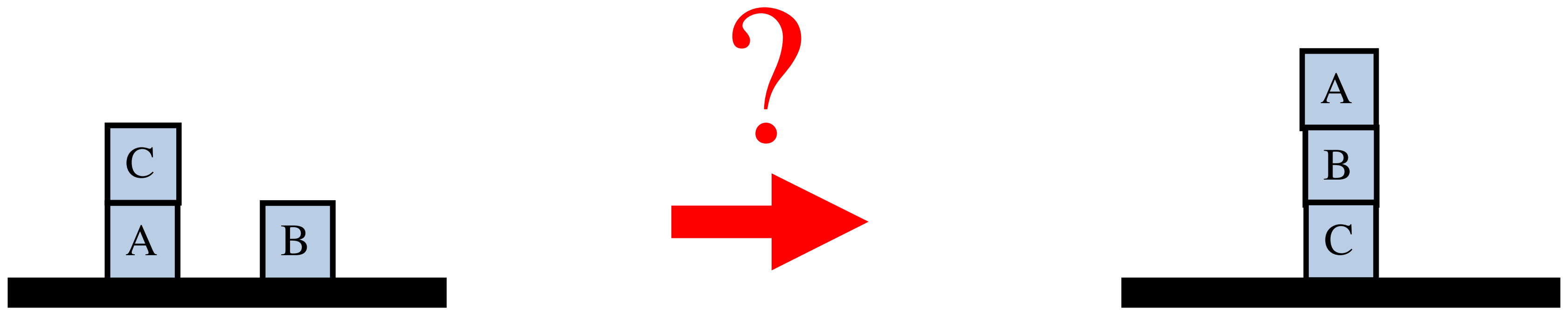
- **<u>The frame problem:</u>** when executing an operator, what is not explicitly changed is considered unchanged.
- **<u>Closed world assumption:</u>** in a state, a term not explicitly mentionned is considered to be *false*.
  - As opposed to the *open world assumption* (ontologies) : *unknown*.

# PDDL: Example of problem

```
(define   (problem blocks-24-1)
          (:domain blocks)
          (:objects X W V U T S R Q P O N M L K J I H G F E D C A B)
          (:init
                    (CLEAR K) (CLEAR I) (ONTABLE C) (ONTABLE O)
                    (ON K F) (ON F T) (ON T B) (ON B G) (ON G R)
                    (ON R M) (ON M E) (ON E J) (ON J V) (ON V N)
                    (ON N U) (ON U H) (ON H C) (ON I A) (ON A P)
                    (ON P Q) (ON Q D) (ON D W) (ON W X) (ON X S)
                              (ON S L) (ON L O) (HANDEMPTY))
          (:goal (and
                    (ON L C) (ON C P) (ON P Q) (ON Q M) (ON M B)
                    (ON B G) (ON G F) (ON F K) (ON K E) (ON E R)
                    (ON R A) (ON A W) (ON W T) (ON T N) (ON N J)
                    (ON J U) (ON U S) (ON S D) (ON D H) (ON H V)
                    (ON V O) (ON O I) (ON I X))))
```


K F T B G R M E J V N U H C
I A P Q D W X S L O
L C P Q M B G F K E R A W T N J U S D H V O I X

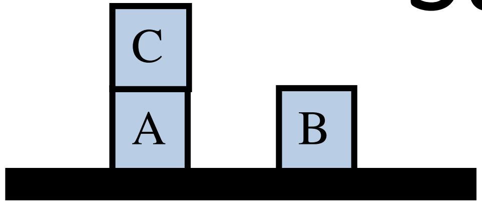
# The anomaly of Gerald Jay Sussman (1/16)

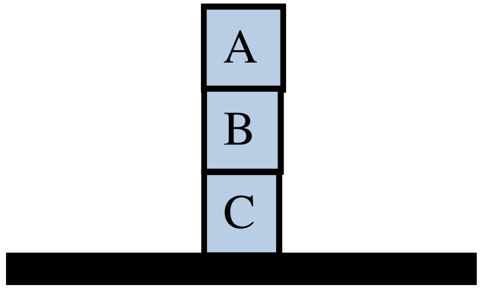


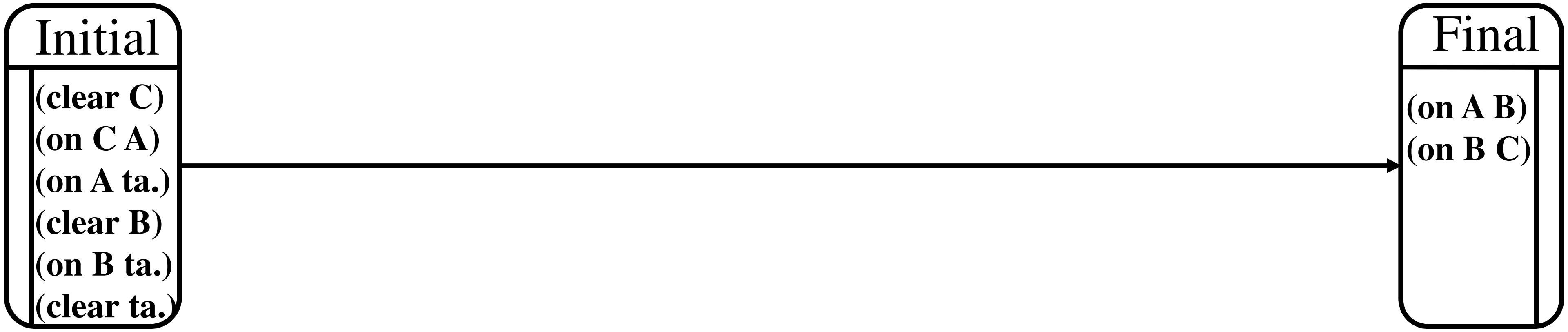
with:

```
puton ?b ?u ?t
(clear ?b)   | (not (on ?b ?u))
(on ?b ?u)   | (=> (<> ?u table)
(clear ?t)   |       (clear ?u))
             | (on ?b ?t)
             | (=> (<> ?t table)
             |       (not (clear ?t)))
```

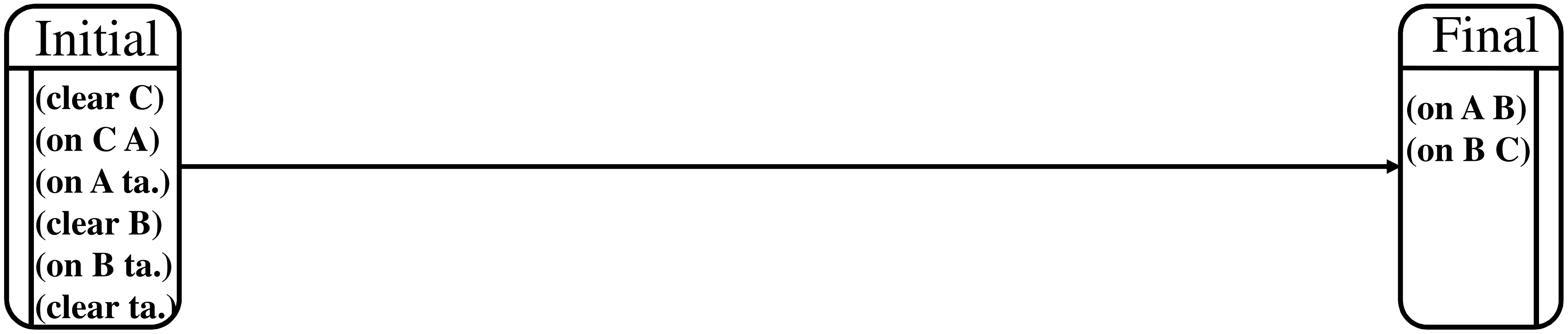
# The anomaly of Gerald Jay Sussman (2/16)



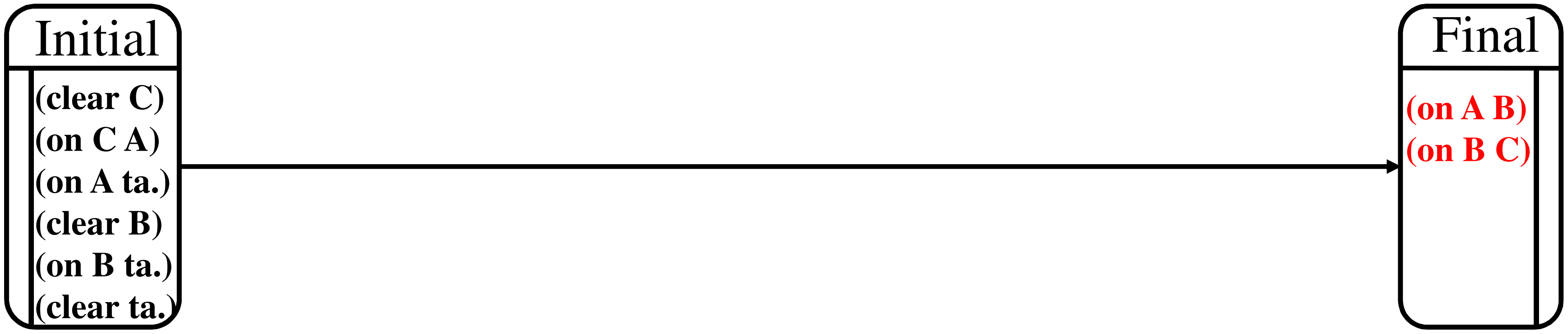




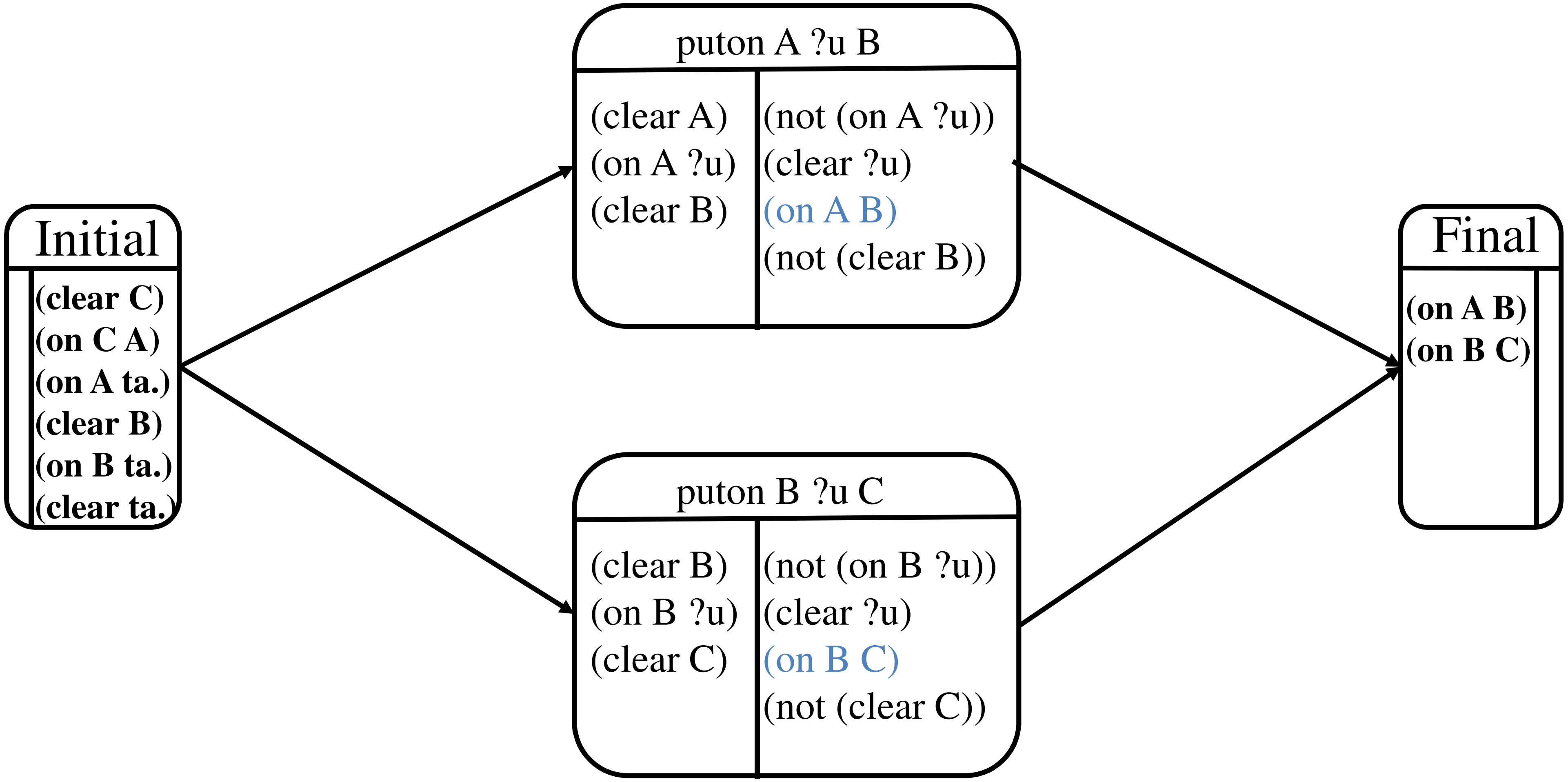
# The anomaly of Gerald Jay Sussman (2/16)



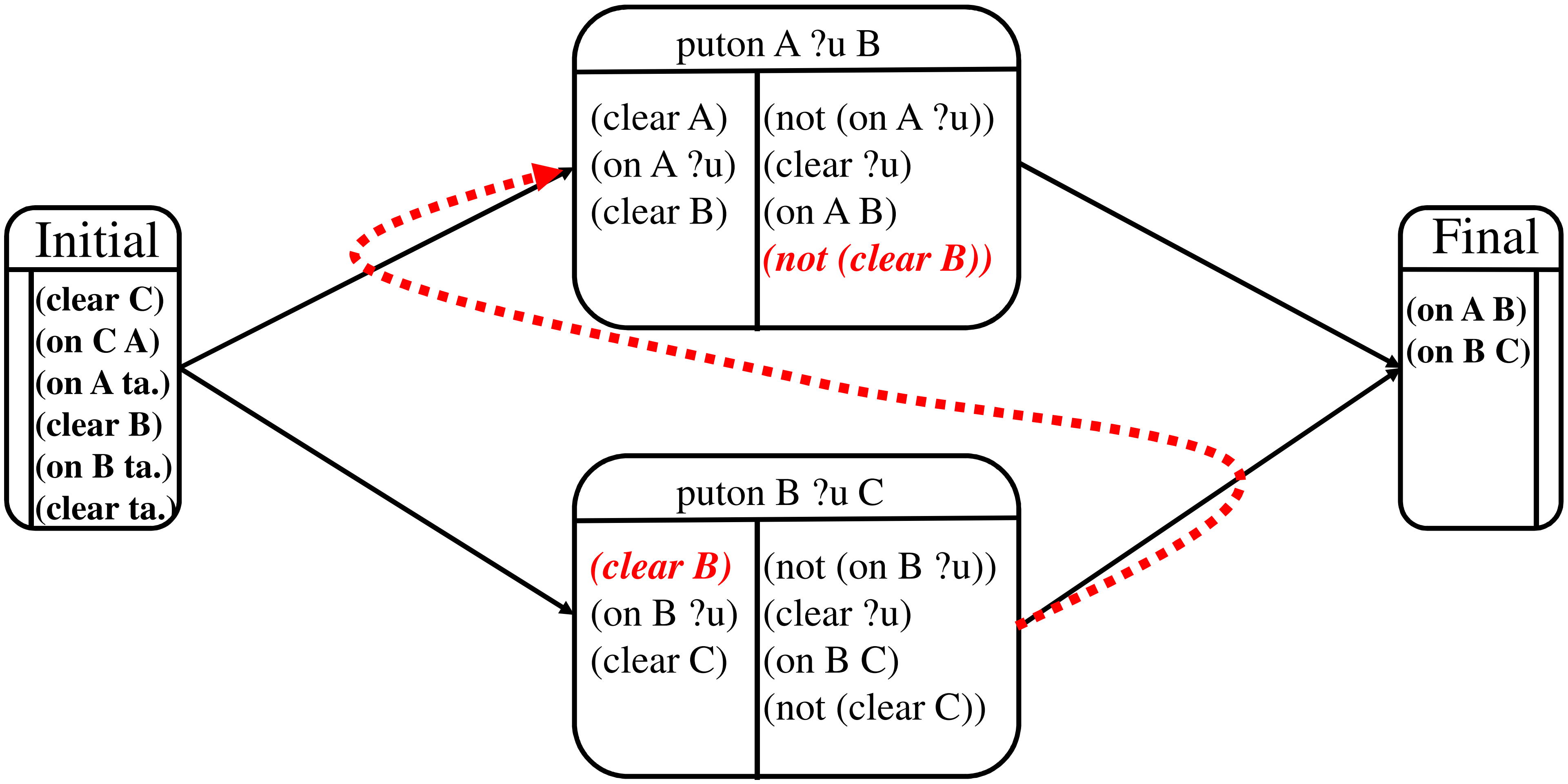
# The anomaly of Gerald Jay Sussman (3/16)



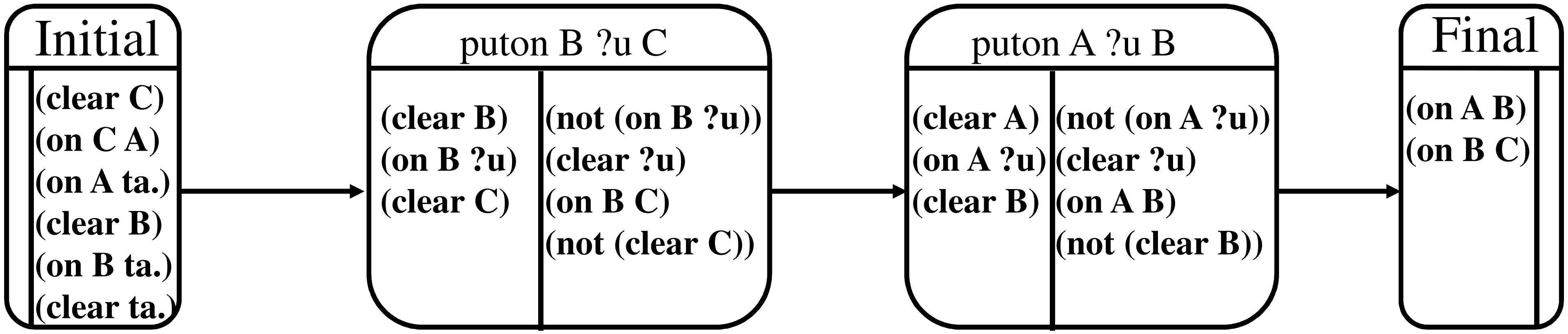
## The anomaly of Gerald Jay Sussman (4/16)



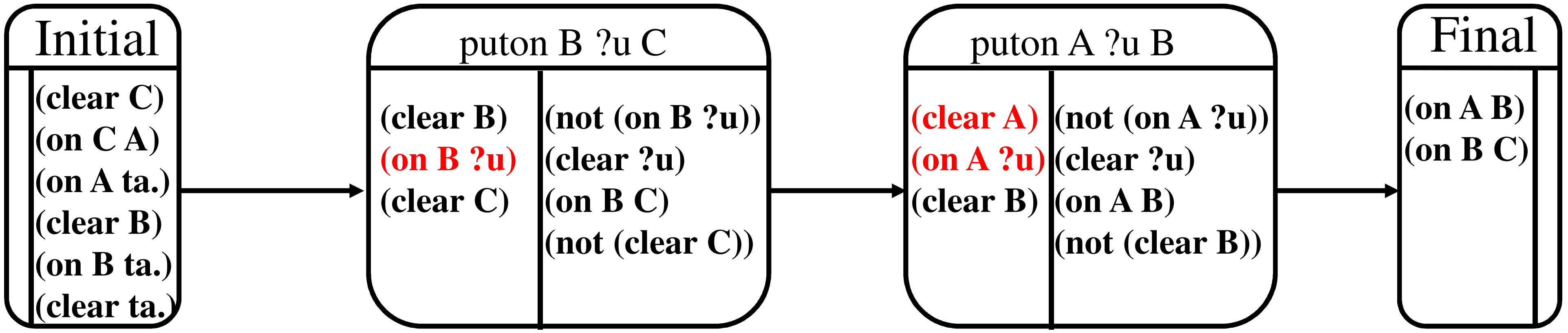
# The anomaly of Gerald Jay Sussman (5/16)



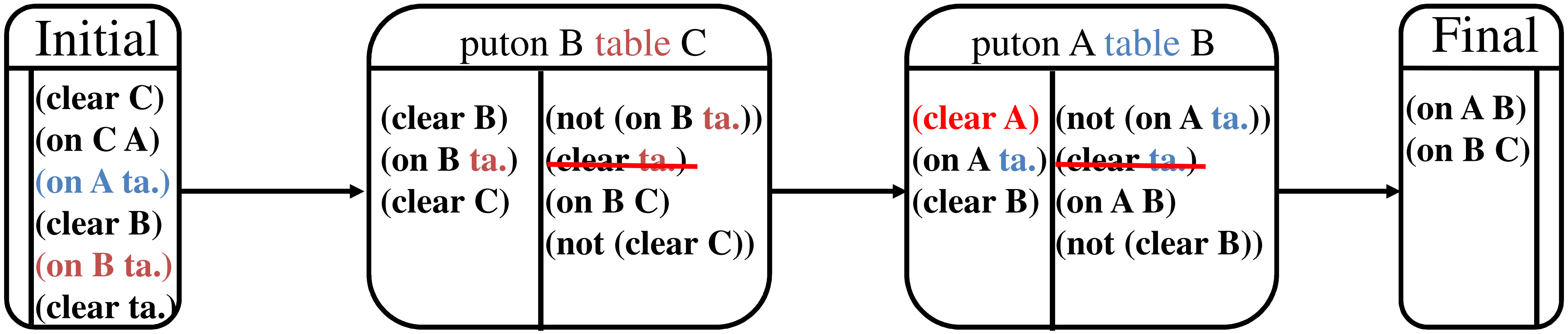
## The anomaly of Gerald Jay Sussman (6/16)



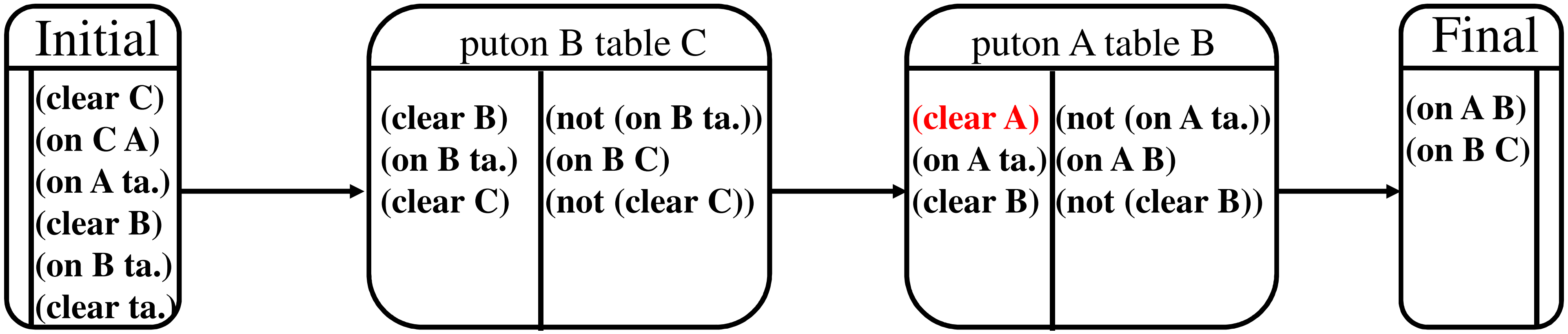
## The anomaly of Gerald Jay Sussman (7/16)



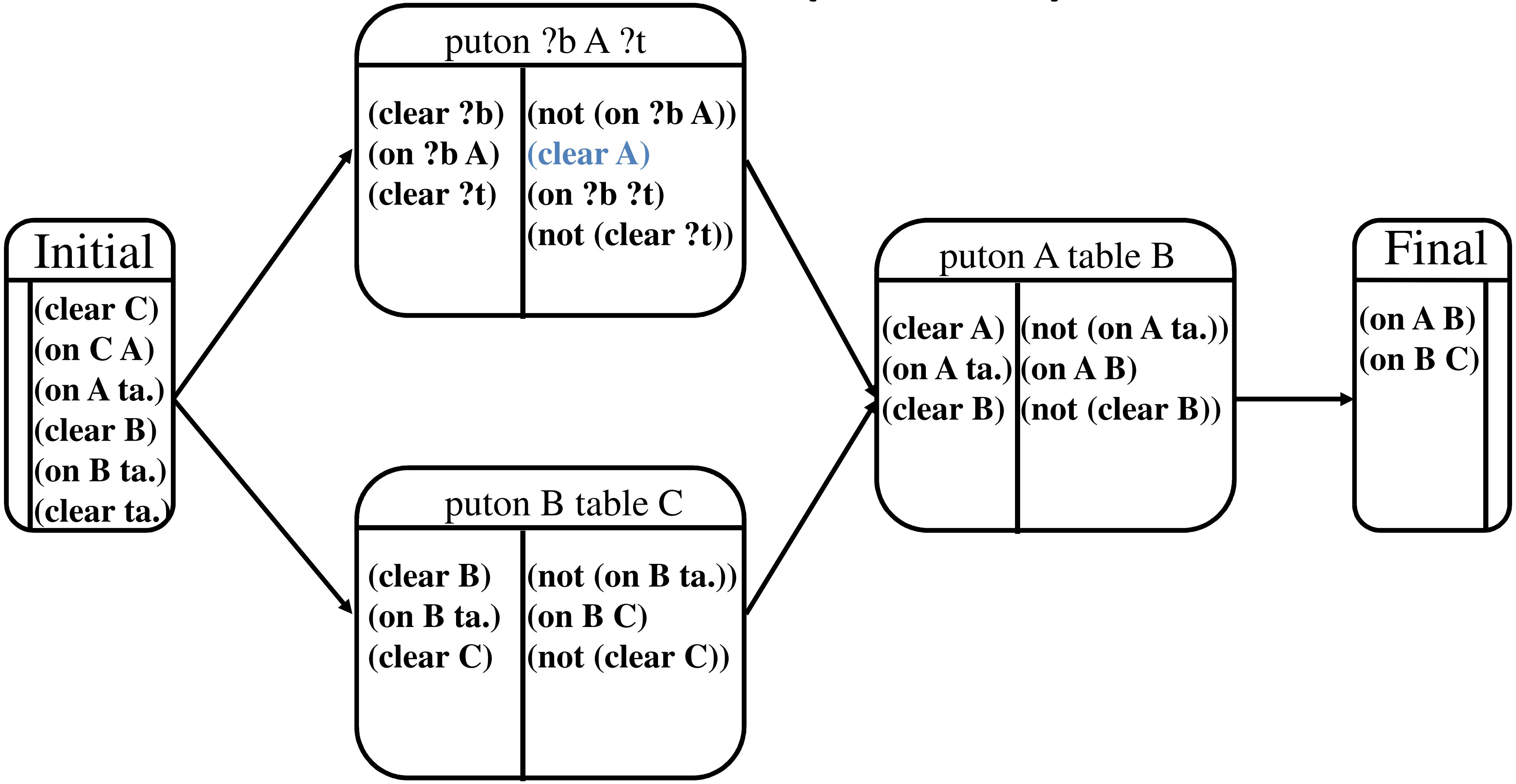
# The anomaly of Gerald Jay Sussman (8/16)



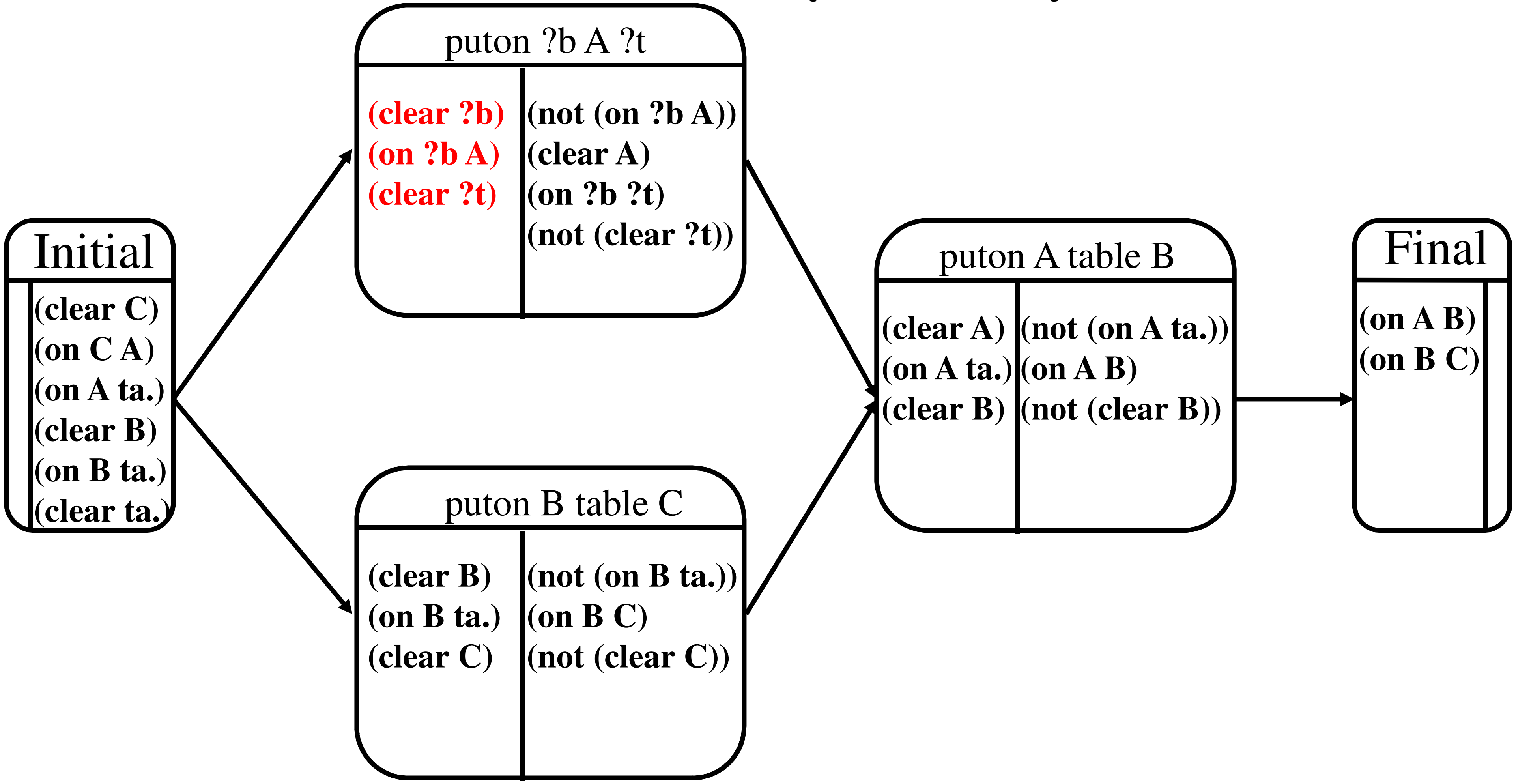
# The anomaly of Gerald Jay Sussman (9/16)



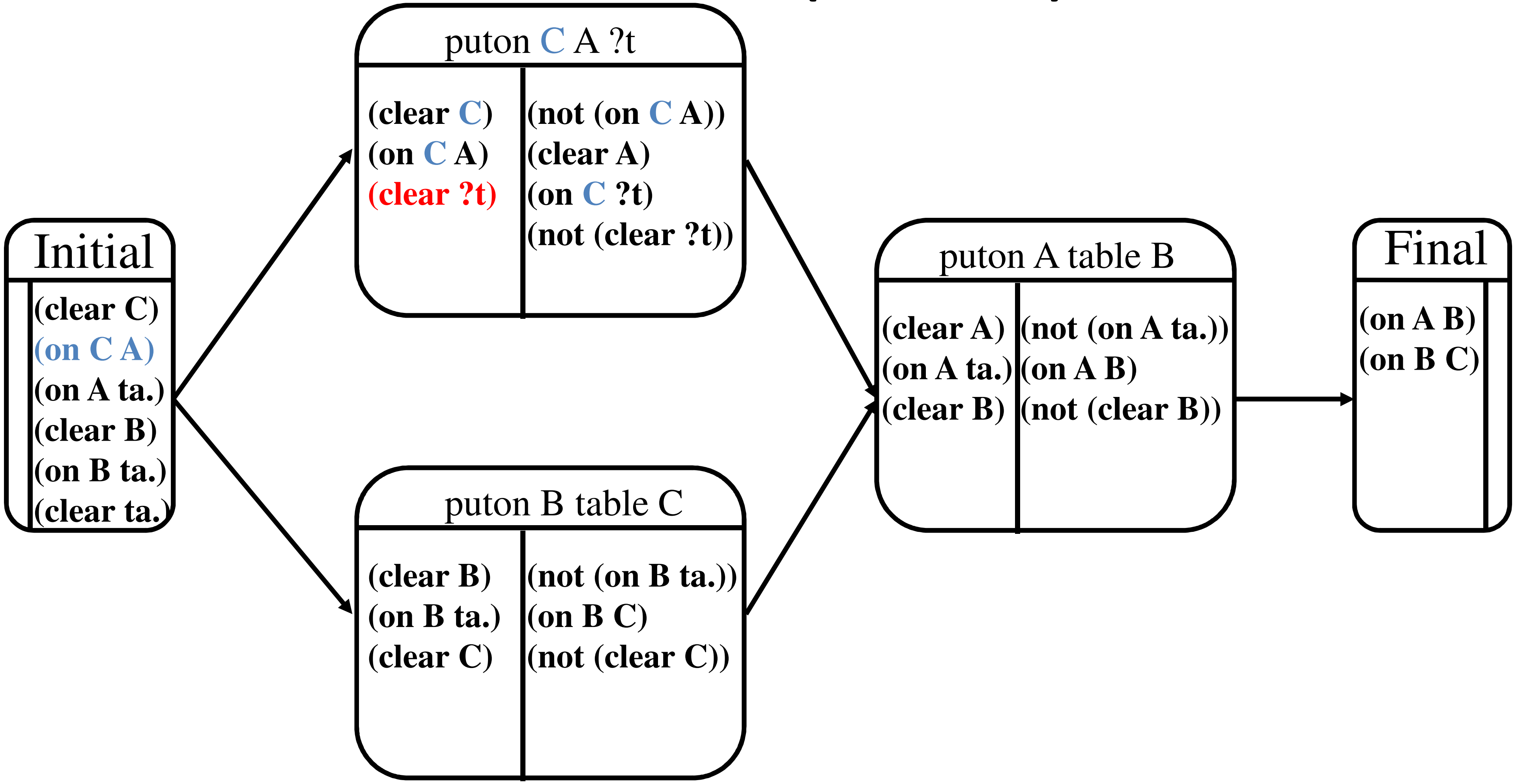
## The anomaly of Gerald Jay Sussman (10/16)



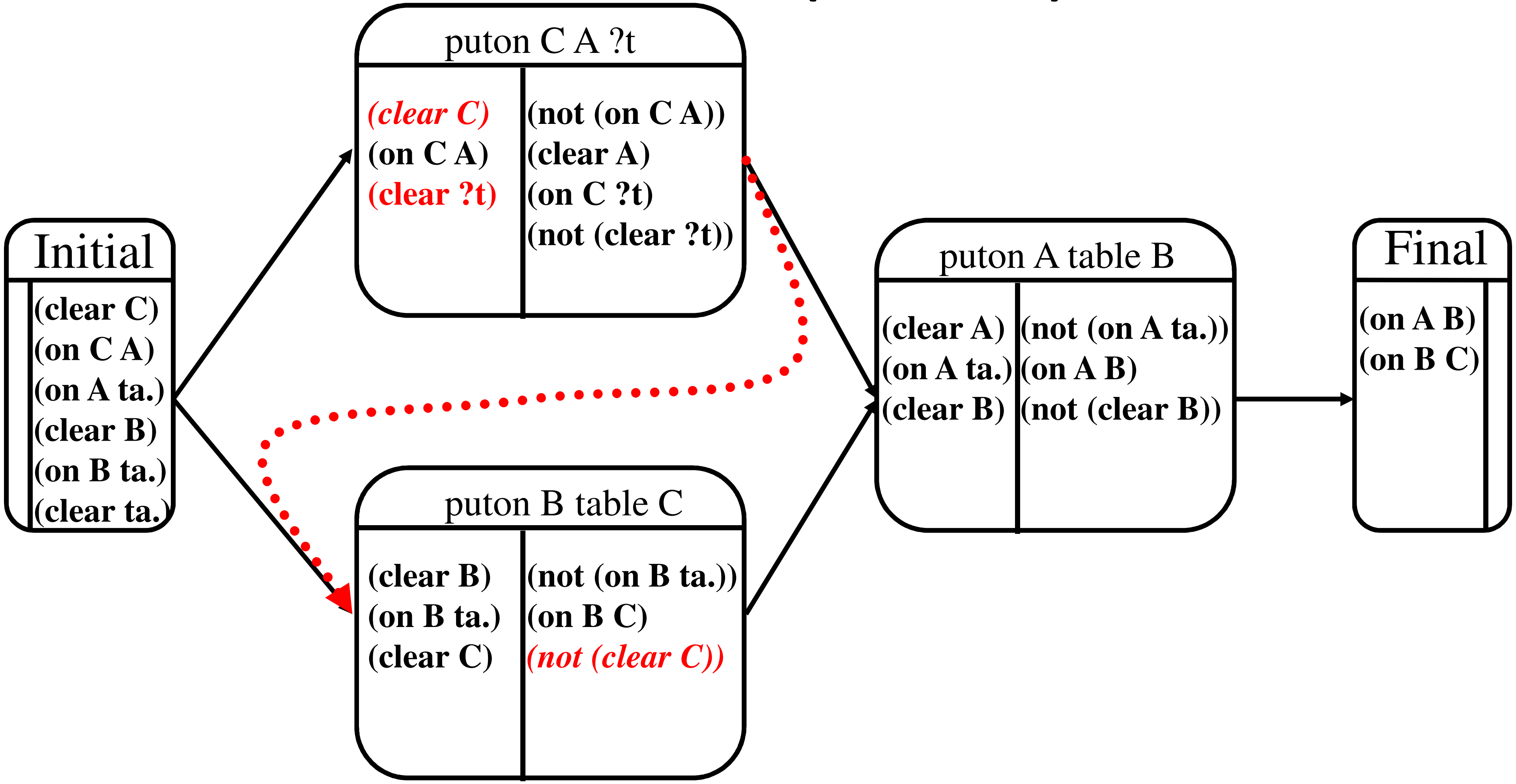
# The anomaly of Gerald Jay Sussman (11/16)



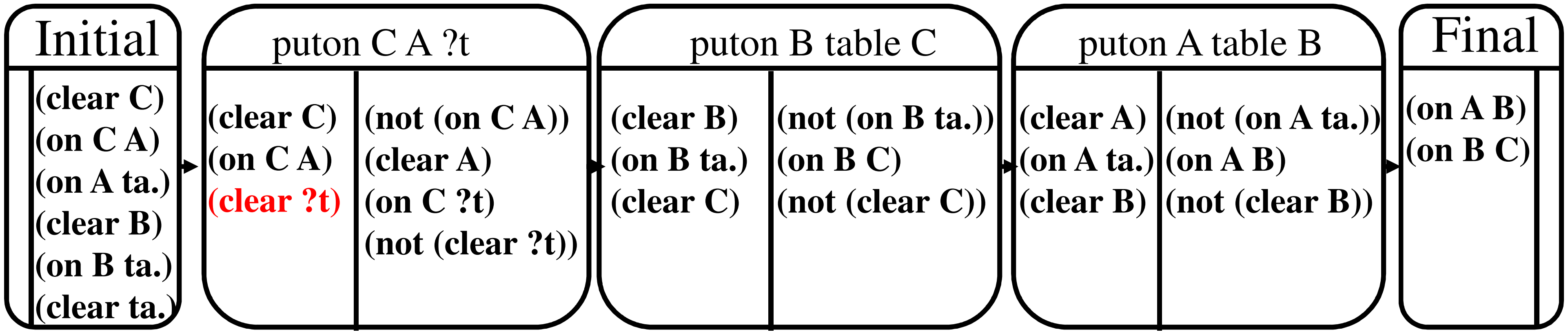
# The anomaly of Gerald Jay Sussman (12/16)



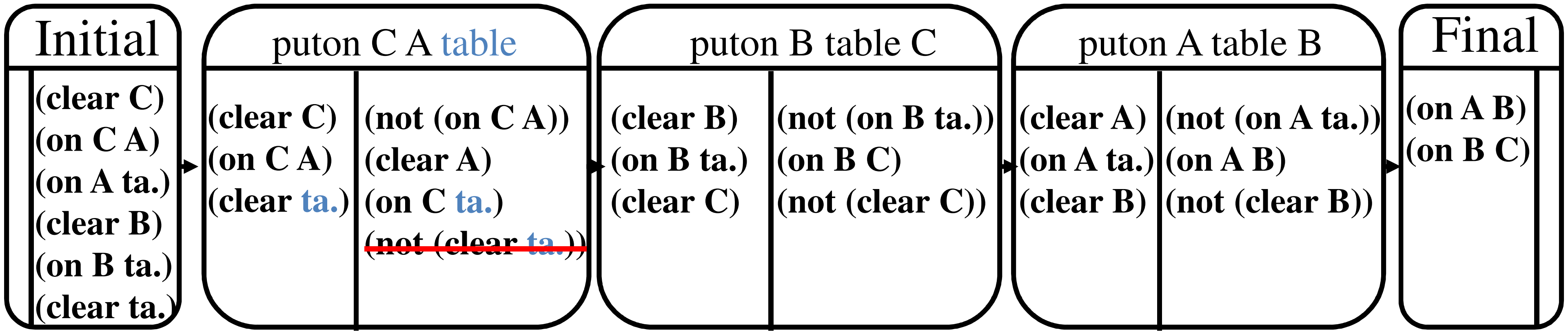
# The anomaly de Gerald Jay Sussman (13/16)



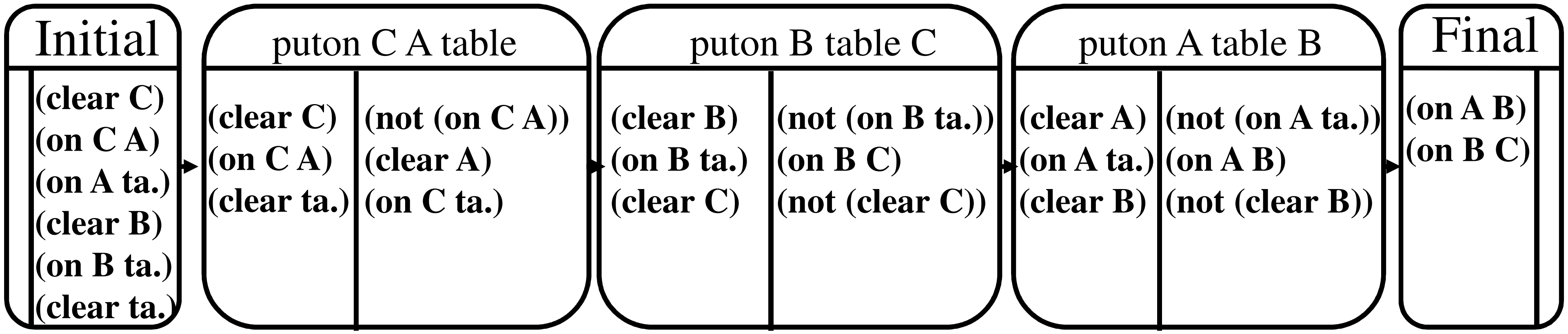
# The anomaly of Gerald Jay Sussman (14/16)



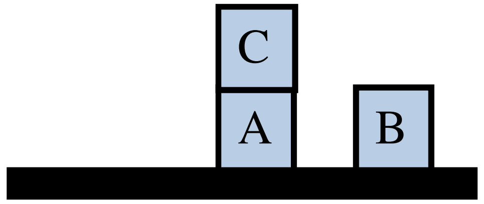
# The anomaly of Gerald Jay Sussman (15/16)



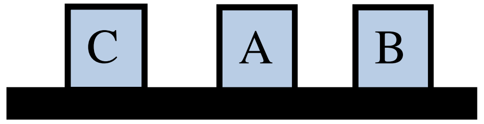
# The anomaly of Gerald Jay Sussman (16/16)



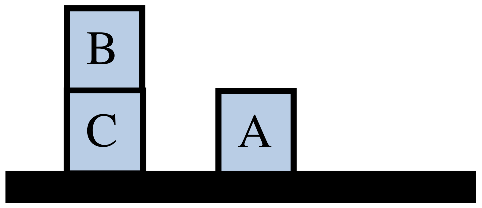
# The anomaly of Gerald Jay Sussman : solution



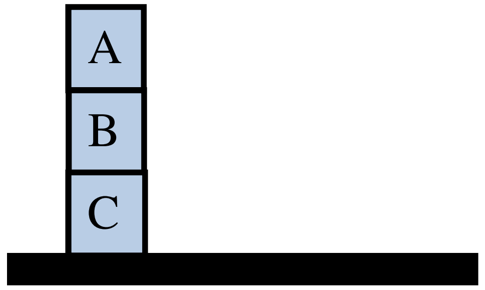
(1)



(2)



(3)

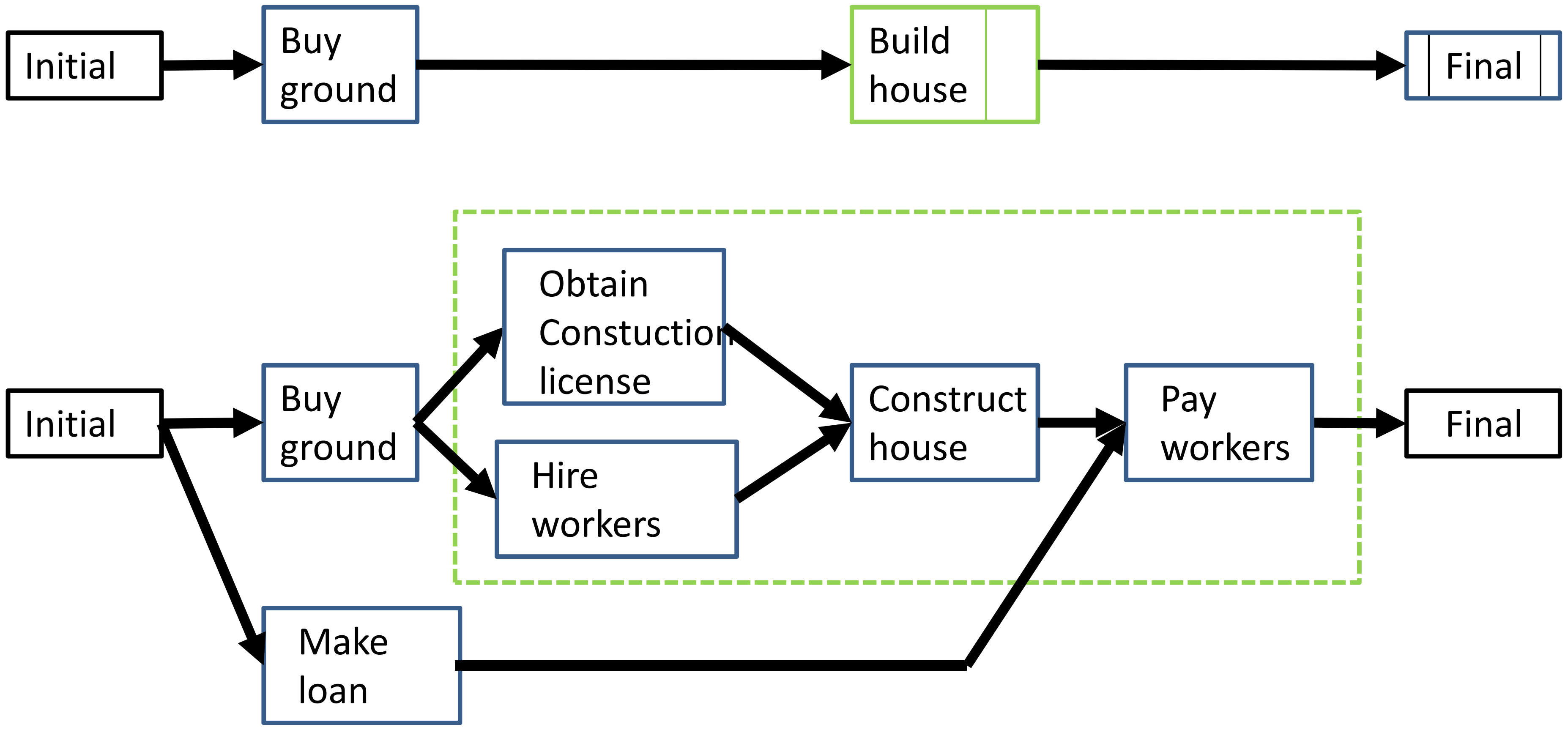


(4)

# Planners …

- Planners using forward search in a state space (Jorg Hoffman, Hector Geffner).
- Planners using backward search in a state space (Malte Helmert).
- Planners using (forward) search in a plan space (Anthony Barrett).
- Planners using evolutionnary algorithms (Marc Schoenauer)
- Planners using temporal logic (Patrick Doherty).
- Planners using constraint programming (Vincent Vidal).
- Planners using SAT solvers (Henry Kautz & Bart Selman, Jussi Rintanen).
- Planner using mixed integer programming (Dana Nau).
- Planner using hierarchical task networks (Dana Nau).

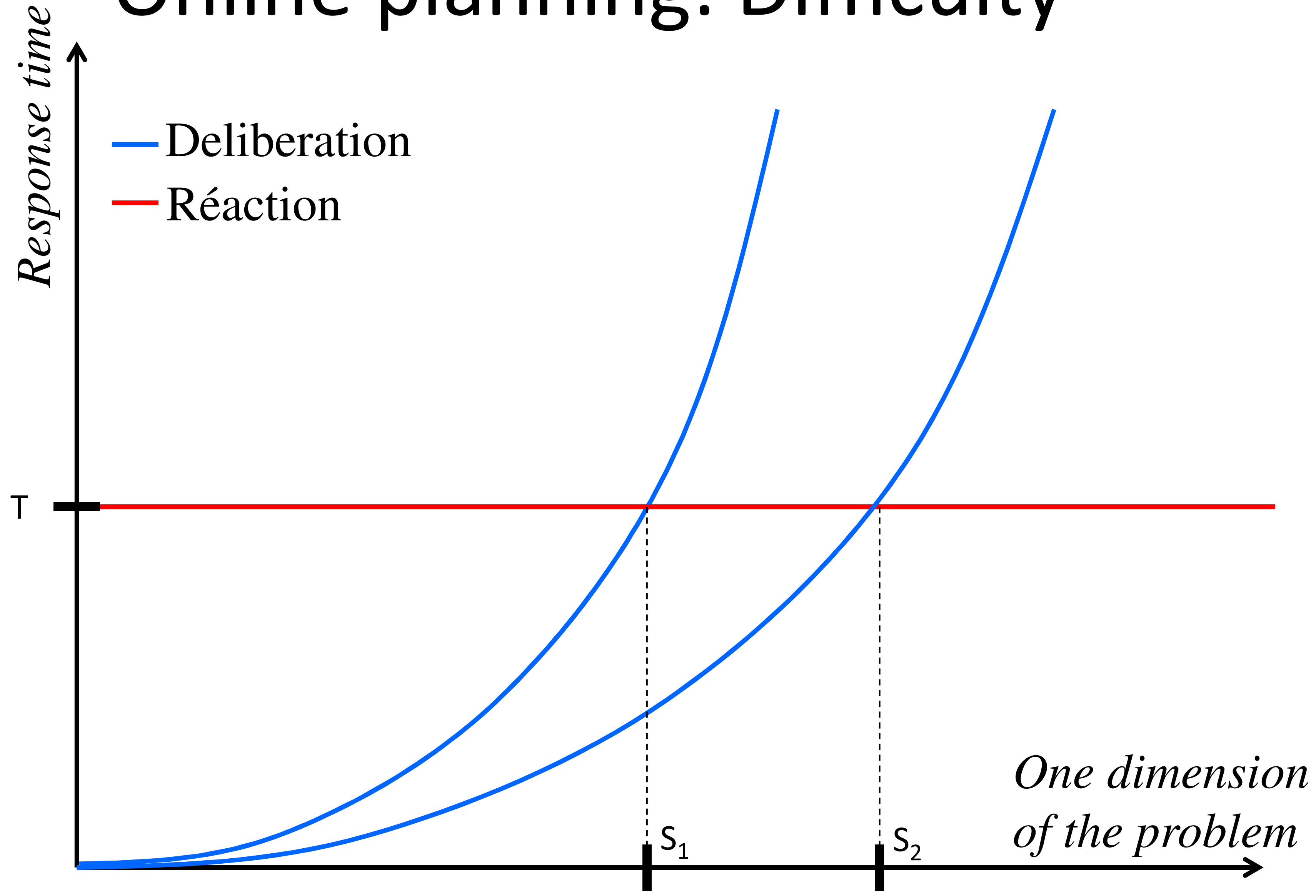
# Hierarchical task networks



## Applications

- Advise a worker to disassemble a car engine (NOAH, Earl Sacerdoti 1974)
- Organize the logistics of the military invasion of Iraq during the first Gulf War (SIPE, David Wilkins, 1980).
- Reactivate the electronics components of a spatial probe cruising around Jupiter (2000).
- Debug a xerox machine.
- Determine the actions of characters in a video game (Eric Jacopin, 2008).
- Interactive story telling (Marc Cavazza, 2010).

# Online planning: Difficulty



# References

- [Weld 94] Daniel Weld, *An Introduction to Least Commitment Planning*, A. I. Magazine, 15(4), pages 27-61, Winter 1994.
- [Russel 2010] Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010, 3rd edition. Chapiter 11.
- [Ghallab et al. 04] Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Mateo, CA, May 04, 635 pages.
- PDDL 3.1. http://ipc.informatik.uni-freiburg.de/PddlExtension
- Conferences :
  - International Conference on Automated Planning and Scheduling (ICAPS). http://www.icaps.org
  - International Joint Conference on A.I. (IJCAI). http://www.ijcai.org
  - European Conference on A.I. (ECAI). http://www.ecai.org
  - National Conference on A.I. (AAAI). http://www.aaai.org
- Journals :
  - A. I. Journal (AIJ). http://www.elsevier.com/wps/find/journaldescription.cws_home/505601/description#description
  - Journal of A.I. Research (JAIR). http://www.jair.org/

## Conclusion

- Action planning consists of finding a sequence of instantiated actions (a plan of operators) which provably leads an initial state to a (final) state containing predefined goals.
  - Difficult because interaction among actions and combinatorial explosion.
- Operators are expressed in the Planning Domain Definition Language (PDDL) and are composed of pre-conditions and post-conditions.
- Several approaches to implement an action planner.
- Planning while executing is online planning: a fast reaction time is required whereas action planning is a combinatorial problem. Cognitive architectures as an approach for that.

# Part II : Constraint Programming

Philippe Morignot
pmorignot@yahoo.fr

## Table of Contents

## General Concepts

- Constraint Programming is a paradigm for solving combinatorial problems.
- Other approaches:
  - Mixed Integer Programming
  - Evolutionary algorithms
  - Search algorithms in a state space (e.g., blind search, heuristic search such as A*).
  - Simulated annealing
  - Taboo algorithm
  - …

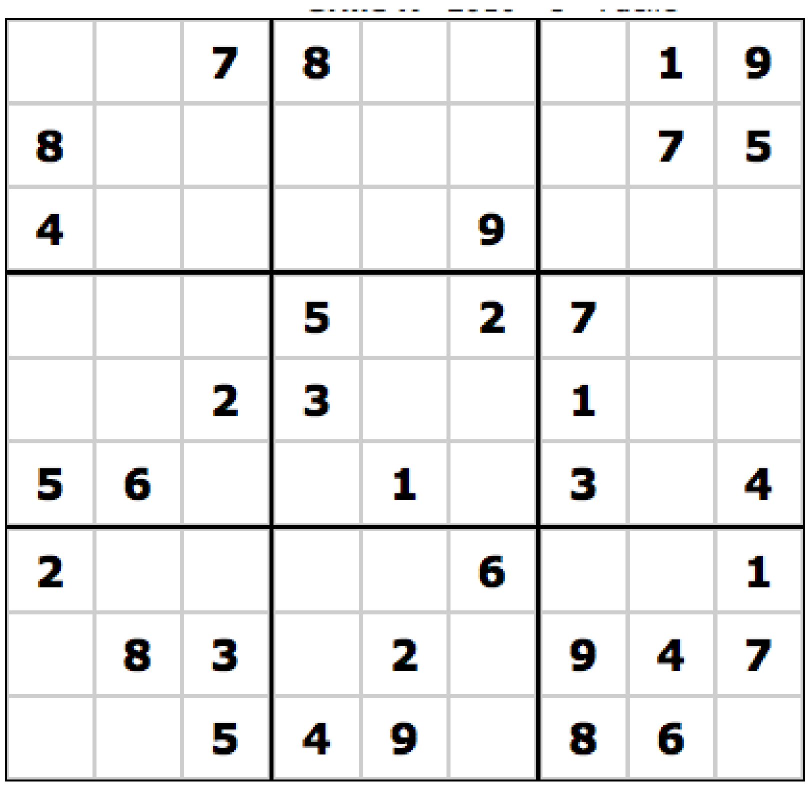
## General Concepts

- A combinatorial problem is a problem …
  - … which can be modelled as entities …
  - … maintaining relations …
  - … and in which a solution must be found:

    the solution to the problem.
- There can be several solutions.
    - Finding one
    - Finding the best one

# General Concepts

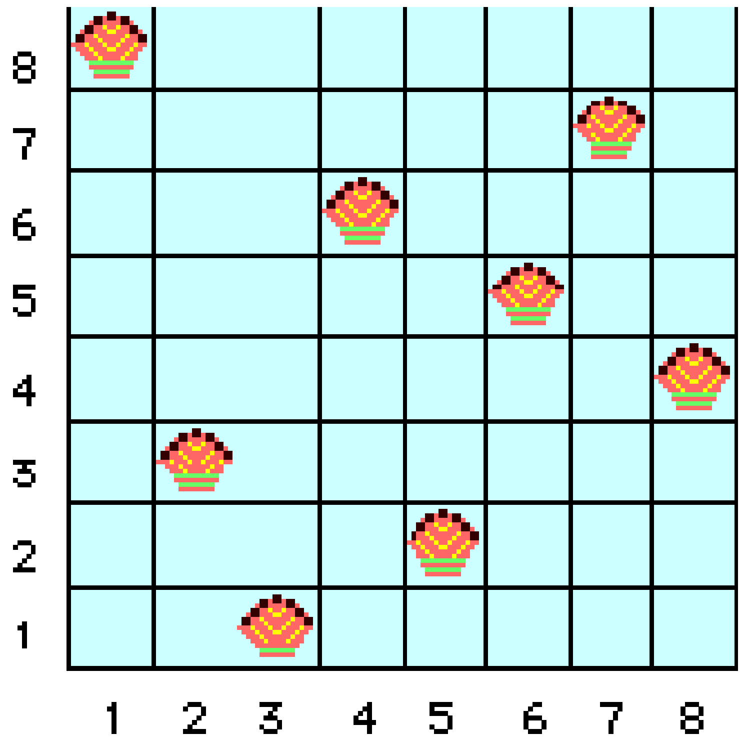
- Example of a combinatorial problem: the sudoku game.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 7 | 8 | | | | 1 | 9 |
| 8 | | | | | | | 7 | 5 |
| 4 | | | | | 9 | | | |
| | | | 5 | | 2 | 7 | | |
| | | 2 | 3 | | | 1 | | |
| 5 | 6 | | | 1 | | 3 | | 4 |
| 2 | | | | | 6 | | | 1 |
| | 8 | 3 | | 2 | | 9 | 4 | 7 |
| | | 5 | 4 | 9 | | 8 | 6 | |

## General Concepts

- Example of a combinatorial problem: the N-queen problem (here, N = 8).



## General Concepts

- Example of combinatorial problem: cryptarithmetics.

```
   SEND
 + MORE
 ---------
  MONEY
```

```
       UN
     DEUX
   + DEUX
   + DEUX
   + DEUX
   --------
     NEUF
```

# General Concepts

- The difficulty: the number of combinations to consider can be gigantic for a real world problem.
  - Example: for the sudoku game, a coarse estimation of the number of combinations is :

    $(8!)^9 \approx 10^{41}$ combinations.
  - For small combinatorial problems, (almost) every algorithm should work …
- Consequence : browsing all these combinations one by one would take a gigantic time, even on the fastest computer.
  - Phenomenon of combinatorial explosion.
  - In the worst case, the number of combinations to consider is an exponential function of the size of one dimension of the data.

# General Concepts

```
while(...) {
   combinaison = nextCombinaison(existant);
   if (qualite(combinaison) > bestQualite) {
      bestQualite = qualite(combinaison);
      bestCombinaison = combinaison;
   }
}
```

**MUCH TOO LONG !!!!**
**(except on small problems)**

## General Concepts

- Idea of constraint programming:
  - Consider the structure of the problem: Decompose the problem as
    - Variables
      - Each variable has a finite domain (variable expressed in extension).
    - Relations among variables (constraints)
      - A constraint must always be satisfied; It reduces the variables' domains.
  - A unique algorithm cleverly uses this model.
    - Heuristics

# General Concepts

- Other example of split between model and solver:
  - Fact base / rule base and inference engine in knowledge-based systems.
  - (Mixed Integer) Linear Programming in Operational Research.
- Consequence:
  - A user writes a model and gives heuristics.
  - The Constraint Programming engine searches for a solution.

## General Concepts

- Examples of problems « solvable » by CP:
  - Assign companies to interns while following their wishes.
  - Optimally plan air traffic (assign flight corridors to aircrafts, optimize flight crew shifts, …)
  - Schedule tasks, so that the makespan is minimized and the resources consumption are minimized.
  - Optimize component location on a mother board.
  - Build a menu both healthy and tasty in a restaurant.
  - …

# General Concepts

- Constraint Programming has been proposed by Jean-Louis Laurière in 1976 in his *Doctorat d'Etat*.
- Seminal Publication:
  - Jean-Louis Laurière, A Language and a Program for Stating and Solving Combinatorial Problems. Artif. Intell. 10(1): 29-127 (1978).
- Packages : CPLEX CP Optimizer from IBM, CHOCO from EMN, GECODE, CHIP from COSYTEC, SICSTUS PROLOG, MINIZINC+ solvers, etc.
- French Association for Constraint Programming: http://afpc.greyc.fr/web/

## Model

- Discrete variables with finite domain:
  - For $i$ from $1$ to $n$, a variable $V_i$
  - For $j$ from $1$ to $n$, a domain $D_j = \{ v_1, v_2, \ldots, v_{f(j)} \}$.
  - For all $i$ from $1$ to $n$, $V_i \in D_i$
- Constraints on these variables:
  - For $k$ from $1$ to $m$, $C_k = ( X_k, R_k)$ with :
    - $X_k = \{ V_{i1}, V_{i2}, \ldots, V_{ik} \}$ // The variables of $C_k$
    - $R_k \subset D_{i1} \times D_{i2} \times \ldots \times D_{ik}$ // The possible values of these // variables, together compatible // with $C_k$

## Vocabulary

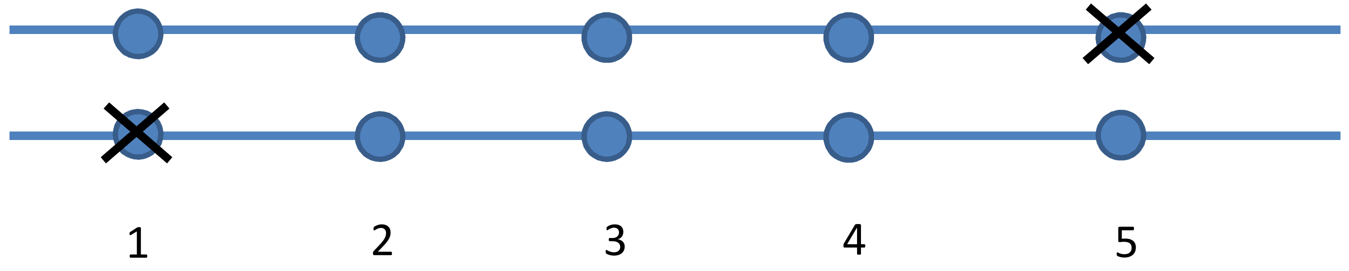
- A CSP (constraint satisfaction problem) might be:
  - Under constrained: too few constraints.
  - Over constrained: too many constraints.
- Given a CSP, one can …:
  - Search for a solution
  - Search for all solutions
  - Search for an optimal solution given some cost function to minimize
  - Prove that there is no solution.

## Constraints

- A constraint can be expressed:
  - In extension: give the sets of possible values of variables
  - Arithmetically: $<$, $\leqslant$, $>$, $\geqslant$, $=$, $\neq$, $+$, $-$, $/$, $*$, …
  - Logically: $\Leftrightarrow$, =>, <=, OR, AND, NOT, …
  - Globally: AllDifferent($x_1$, $x_2$, …, $x_n$), Geost($f_1$, $f_2$, …, $f_n$), ...
- A constraint can be:
  - Hard: a constraint must always be satisfied.
  - Soft: a constraint is sometimes satisfied and sometimes violated, given a criterion
- A contraint can be:
  - Unary. Example : $x \in [1, 5]$
  - Binary. Example : $x < y$
  - N-ary. Example : AllDifferent($V_1$, $V_2$, …, $V_n$)

# Constraints

- Example of hard constraints:
  - $x \in [1, 5]$ ; $y \in [1, 5]$ ; $x < y$
  - x :
  - y :



- Example of soft constraints:
  - In a scheduling problem,

    $Y = \#(t_i < \text{deadline}_i)$ and maximize Y

## Constraints
## Global constraints

- **AllDifferent**($\boldsymbol{V_1}, \boldsymbol{V_2}, \ldots, \boldsymbol{V_n}$)
  - The variables $\boldsymbol{V_i}$ must be all different.
  - Logically equivalent to:

$$\begin{aligned} V_1 \neq V_2 \wedge V_1 \neq V_3 \wedge \ldots \wedge V_1 \neq V_n \wedge \\ V_2 \neq V_3 \wedge \ldots \wedge V_2 \neq V_n \wedge \\ \ldots \wedge \\ V_{n-1} \neq V_n \end{aligned}$$

- <u>Property:</u> if there are **m** variables in AllDiff, and **n** distinct values together possible, and if **m** > **n**, then the constraint cannot be satisfied.

## Example of model

- A model for the sudoku game:
  - A variable is an empty cell in the grid.
  - A domain is the set of integers from 1 to 9
    - If the cell already includes a number, it appears as a constant in the constraint.
  - Constraints:
    - The variables of a small grid are all different and different from constants.
    - The variables of a row are all different.
    - The variables of a column are all different.

# Example of model

- 1st model for the N-queen problem (here, N = 8) :
  - A pair of variables ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$) per queen ***i***. The queen ***i*** is located on colum $\boldsymbol{x_i}$ and on row $\boldsymbol{y_i}$
  - The domain of $\boldsymbol{x_i}$ is [1, 8]
  - The domain de $\boldsymbol{y_i}$ is [1, 8]
  - Constraints:
    - $x_i \neq x_j$ // Different columns
    - $y_i \neq y_j$ // Different rows
    - $x_i + y_i \neq x_j + y_j$ // Different 1st diagonal
    - $x_i - y_i \neq x_j - y_j$ // Different 2nd diagonal

## Example of model

- 2nd model for the N-queen problem (here, N = 8) :
  - The variable $\boldsymbol{x_i}$ is the row of the ***i***-th column on which the i-th queen is located.
  - The domain of $\boldsymbol{x_i}$ is [1, 8]
  - Constraints:
    - The constraints on columns are satisfied by construction.
    - $x_i \neq x_j$ // Different rows
    - $x_i + i \neq x_j + j$ // Different 1sr diagonals
    - $x_i - i \neq x_j - j$ // Different 2st diagonals

# Example of model

- <u>3rd model for the N-queen problem (N = 8) :</u>
  - The cells of the grid are numbered from 1 to 64.
  - The variable $\boldsymbol{x_i}$ is the index of the cell in this numbering at which queen i is located.
  - Constraints:
    - $x_i / 8 \neq x_j / 8$ // Different rows
    - $x_i \% 8 \neq x_j \% 8$ // Different columns
    - Constraints on the 1e diagonal
    - Constraints on the 2e diagonal

# Example of model

- <u>Cryptarithmetics: SEND + MORE = MONEY</u>
- The model:
  - Variables : S, M $\in$ [1, 9] ; E, N, D, O, R, N, Y $\in$ [0, 9]
  - Constraints :
    - $D + E = Y + 10 * R1$
    - $N + R + R1 = E + 10 * R2$
    - $E + O + R2 = N + 10 * R3$
    - $S + M + R3 = O + 10 * M$
    - Secondary variables : R1, R2, R3 $\in$ [0, 1]

## Vocabulary

- An **assignment** is the fact of assigning a variable to a value of its domain.
  - Variable $V_i$ is assigned to its value $v_{i,j}$ : $D_i = \{ v_{i,j} \}$
- An **assignment** of variables to values is :

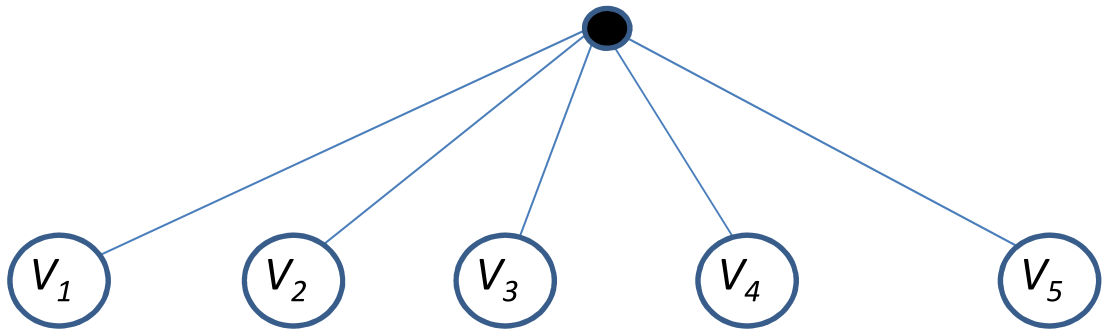
  $A = \{(V_{i1}, v_{i1}), (V_{i2}, v_{i2}), \ldots, (V_{ik}, v_{ik})\}$
- An assignment can be :
  - **total** : every variable has a value ($k = n$)
  - **partial** : some variables have a value, but not all ($k < n$).
- An assignment $A$ is **consistant** iff it does not violate any constraint $C_k$.
- A **solution** to a CSP is a total consistant assignment of variables.
- Some CSPs require to maximize an objective function $f$ (or minimize a cost function, equivament to an objective one, by adding a minus « - » sign).

## Algorithms

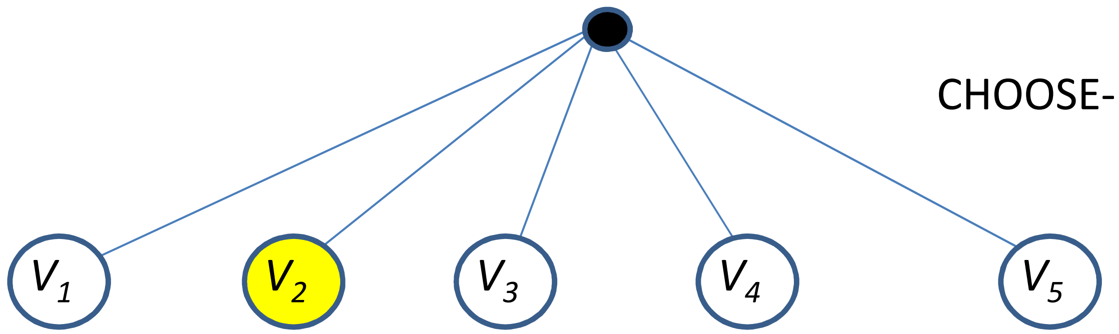
- Commutativity :
  - A problem is commutative iff the order of application of actions has no effect on the result.
- A CSP is commutative : when values are assigned to variables, the same partial assignment is reached whatever the order is.
- Consequence: we can assign variables one after the other, as needed.

## Algorithms : backtrack (1 / 14)

## Algorithms : backtrack (2 / 14)

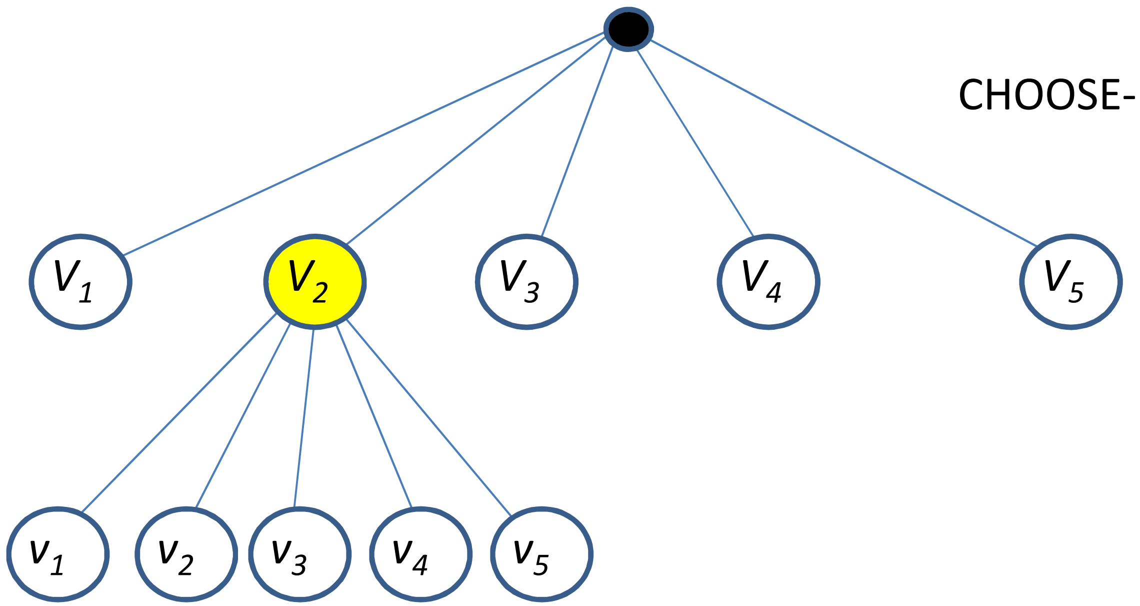


# Algorithms : backtrack (3 / 14)



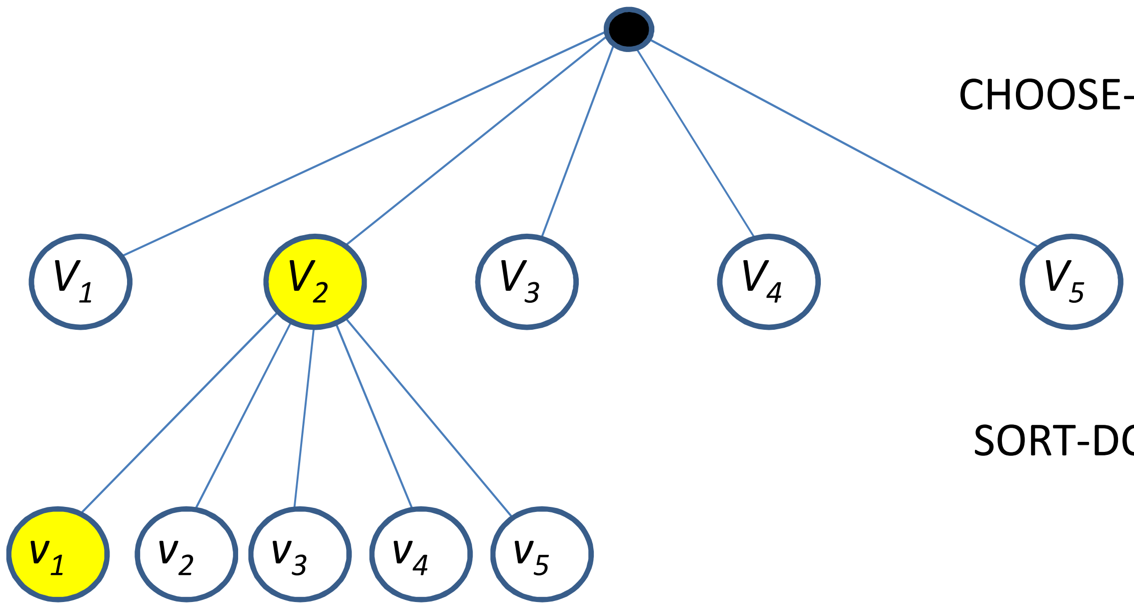
CHOOSE-UNASSIGNED-VARIABLE

# Algorithms : backtrack (4 / 14)



CHOOSE-UNASSIGNED-VARIABLE

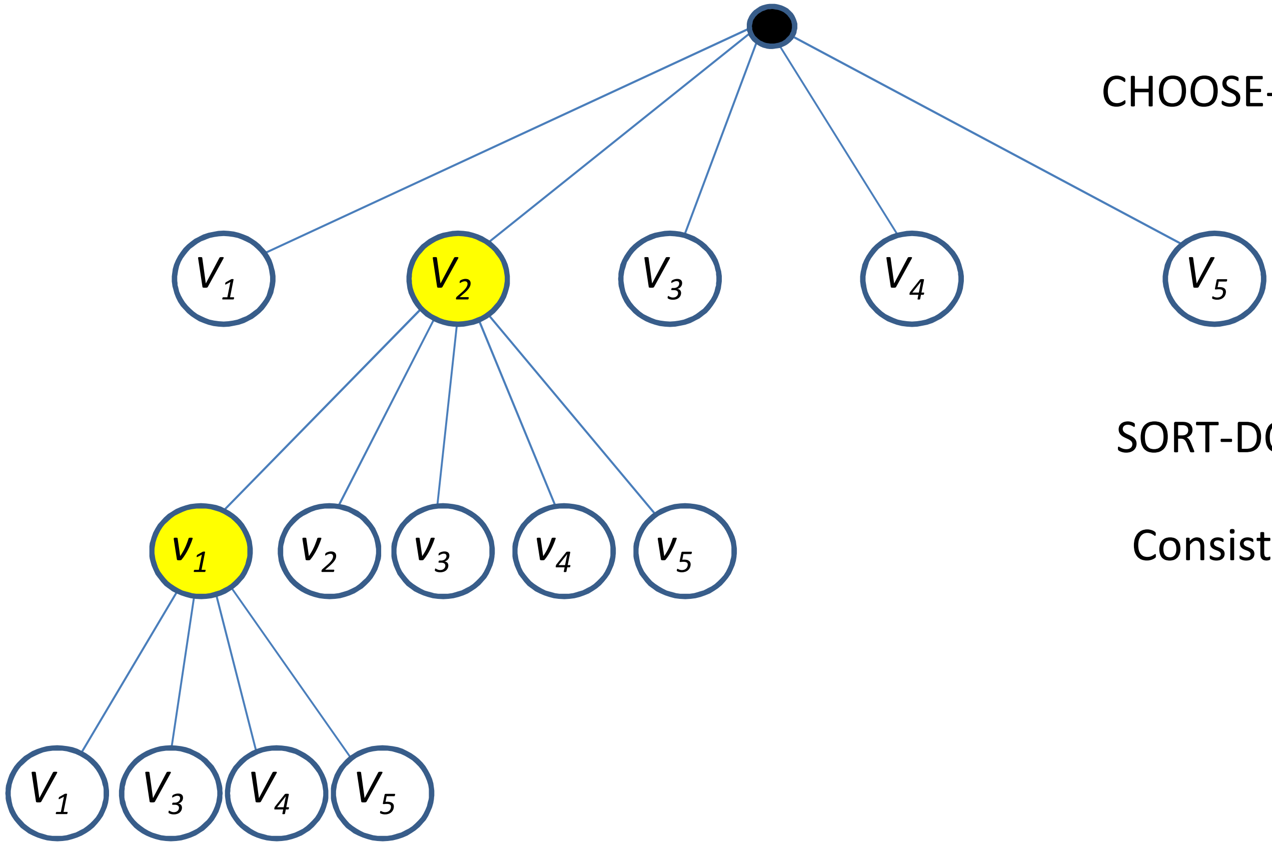
# Algorithms : backtrack (5 / 14)



CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

# Algorithms : backtrack (6 / 14)

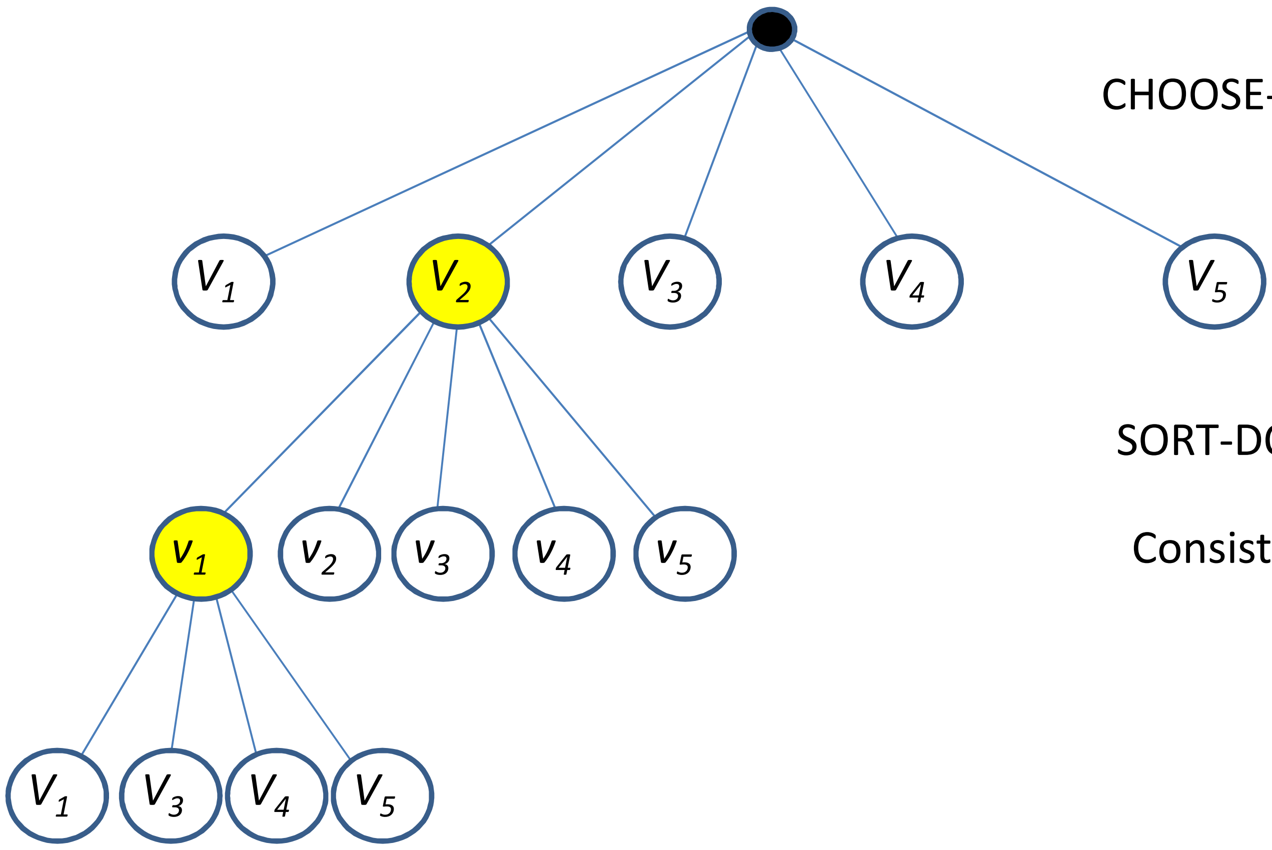


CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

# Algorithms : backtrack (7 / 14)



CHOOSE-UNASSIGNED-VARIABLE

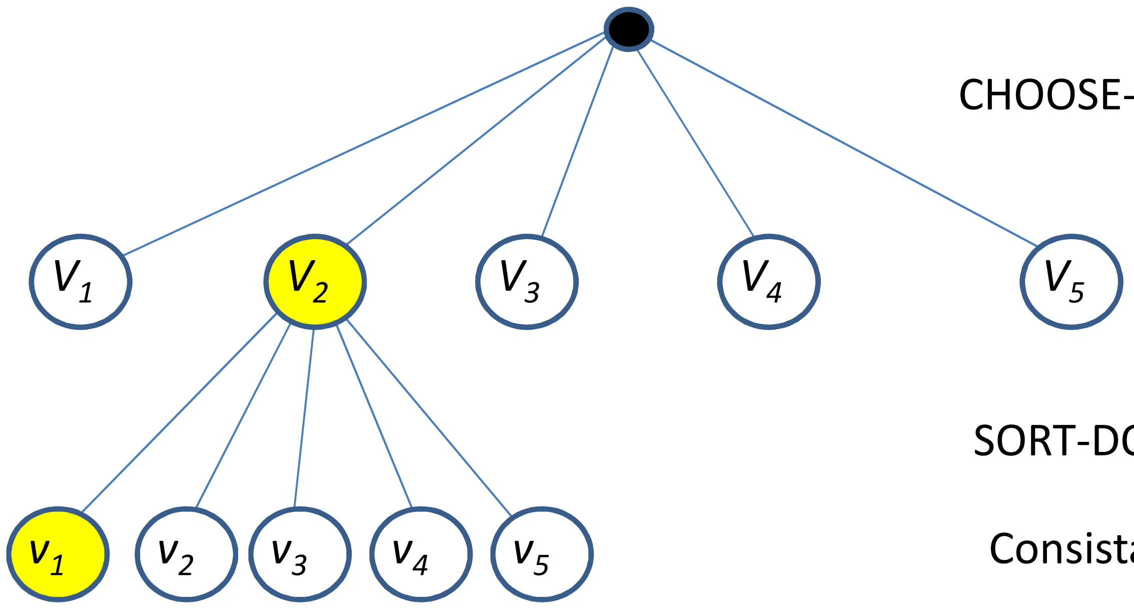
SORT-DOMAIN-VALUES

Consistancy checks

**One domain becomes empty !**

BACKTRACK

# Algorithms : backtrack (8 / 14)

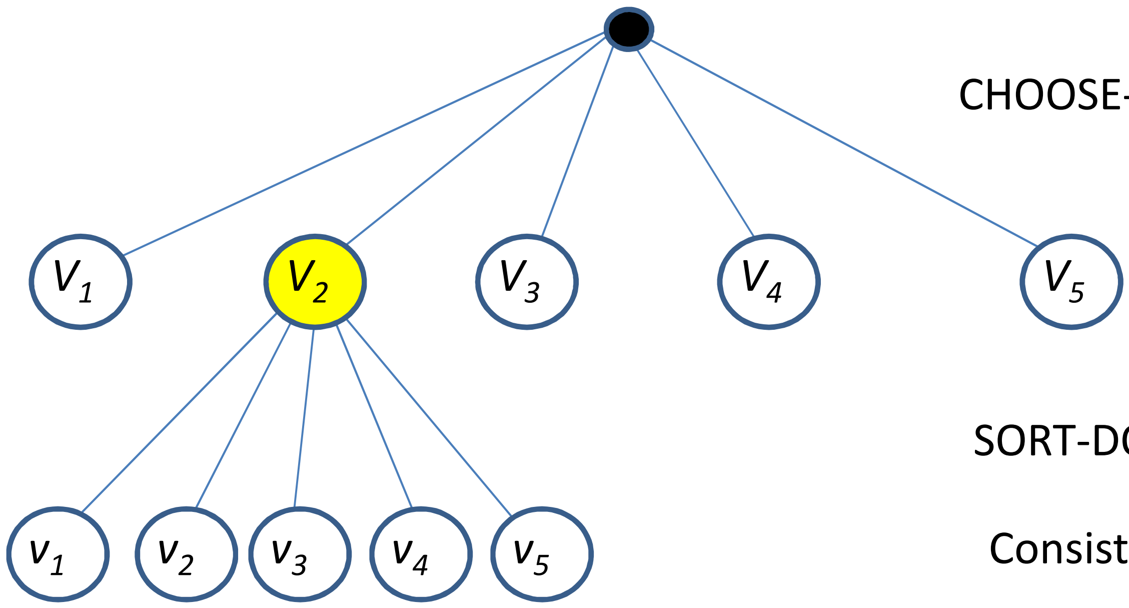


CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

# Algorithms : backtrack (9 / 14)

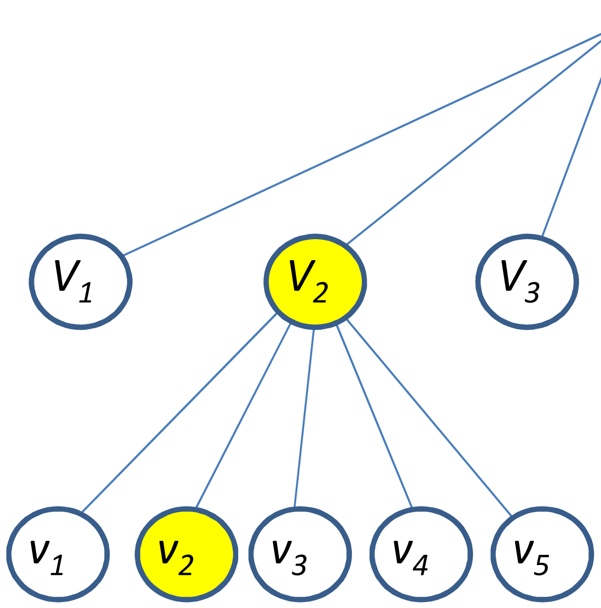


CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

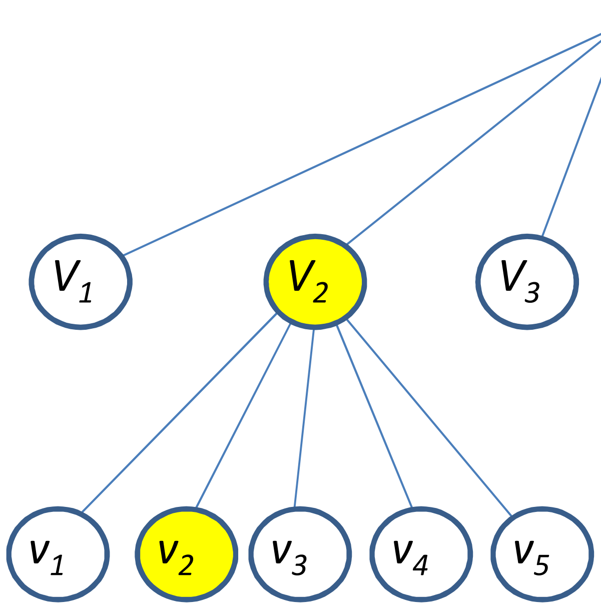
# Algorithms : backtrack (10 / 14)



CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

# Algorithms : backtrack (11 / 14)

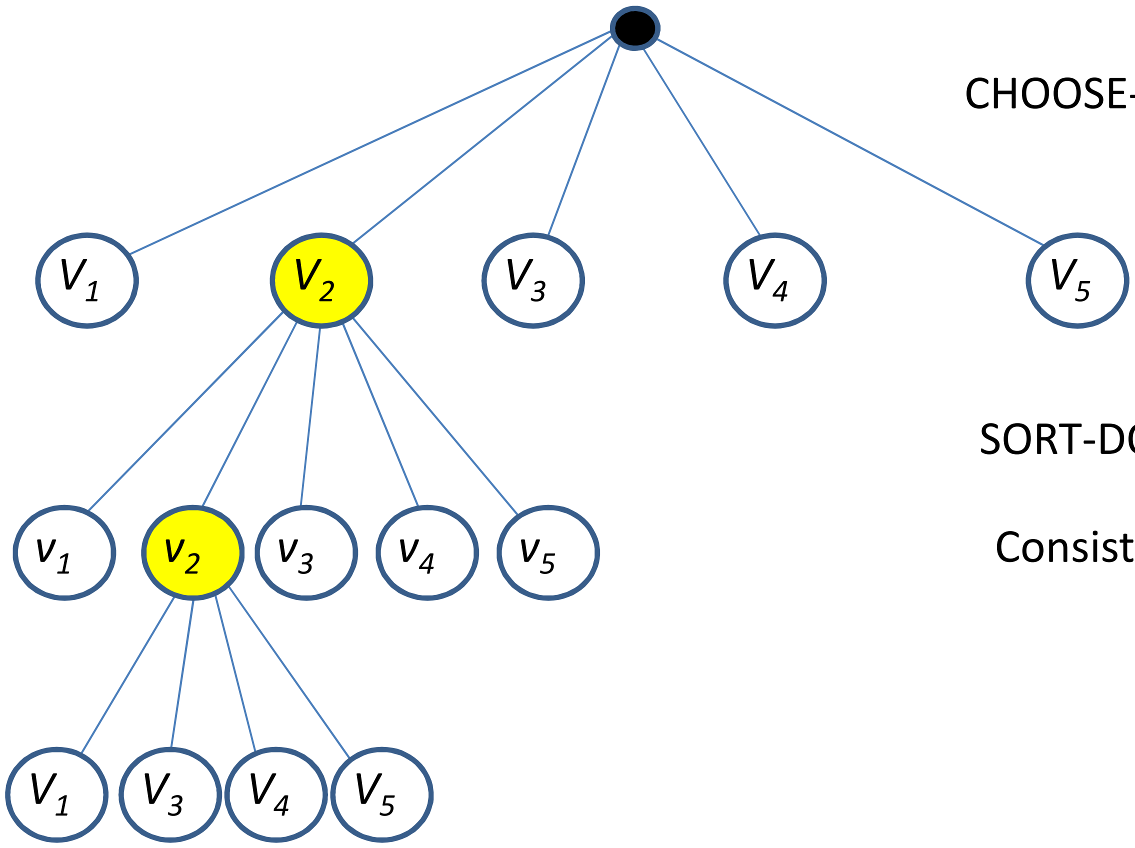


CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

# Algorithms : backtrack (12 / 14)

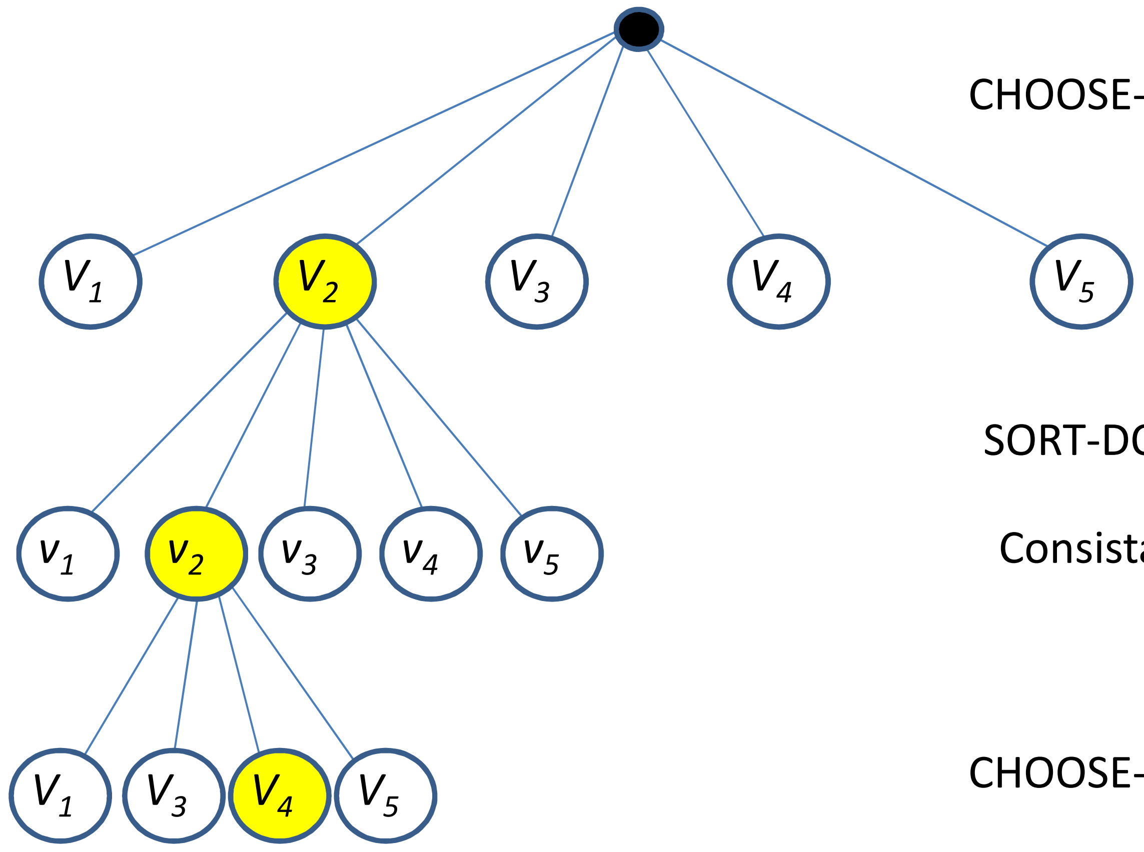


CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

# Algorithms : backtrack (13 / 14)



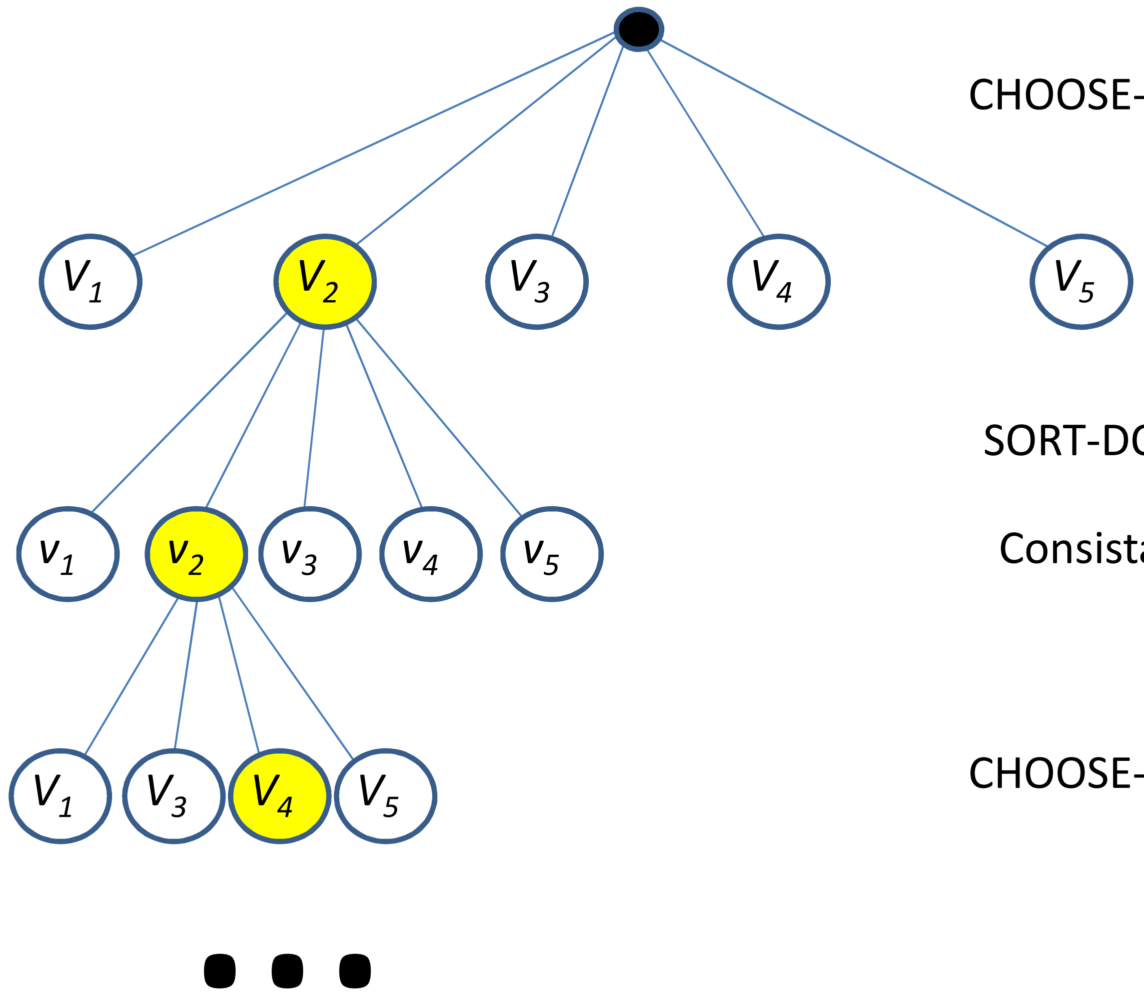
CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

CHOOSE-UNASSIGNED-VARIABLE

# Algorithms : backtrack (14 / 14)



CHOOSE-UNASSIGNED-VARIABLE

SORT-DOMAIN-VALUES

Consistancy checks

CHOOSE-UNASSIGNED-VARIABLE

## Heuristics (1 / 2)

- A heuristics makes a choice better than randomly.
  - Expressed in terms of variables, domains and constraints.
  - Or can be based on exogeneous information (e.g., the application domain of the CSP).
  - Prototype in C++ : int heuristics(Assignment* assignment, Csp* csp);

- **<u>Heuristics on variables :</u>** CHOOSE-UNASSIGNED-VARIABLE()
  - Static / dynamic.
  - **First-fail** : choose the variable with the smallest domain.
  - **Smallest** : choose the variable with the smallest value in its domain.
  - **Constraint**: choose the variable linked to the maximum number of constraints.
  - …

## Heuristics (2 / 2)

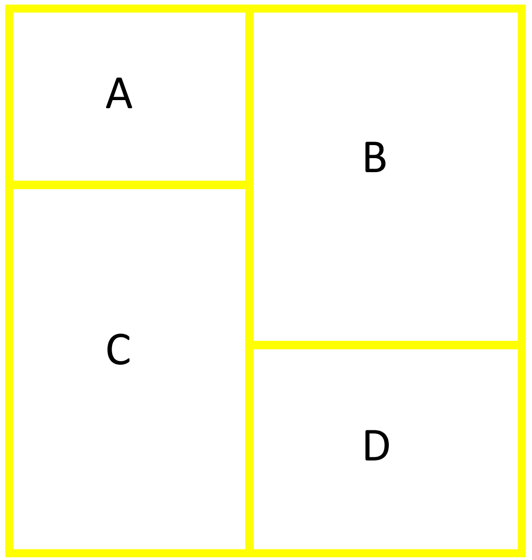
- **Heuristics on values :** SORT-DOMAIN-VALUES()
  - Static / dynamic
  - **Min** : assign a variable to its minimum value
  - **Max** : assign a variable to its maximum value
  - **Median** : assign a variable to its median value; or different from its median value (middle-out)
  - **Split** : constrain the variable to its lower/upper half domain
  - **Regret** : assign the variable to the value which removes the least number of values to other variables.
  - …
- **Search strategy :** additional constraints which orient search.

## Filtering

- What does the assigment of a variable imply for other variables ?
- **<u>FORWARD-CHECKING:</u>** each time a variable $\boldsymbol{V_i}$ is instantiated, consider variables $\boldsymbol{V_j}$ connected to $\boldsymbol{V_i}$ by a constraint $\boldsymbol{C_k}$, and remove from the domain of variable $\boldsymbol{V_j}$ the values which are inconsistant with constraint $\boldsymbol{C_k}$.

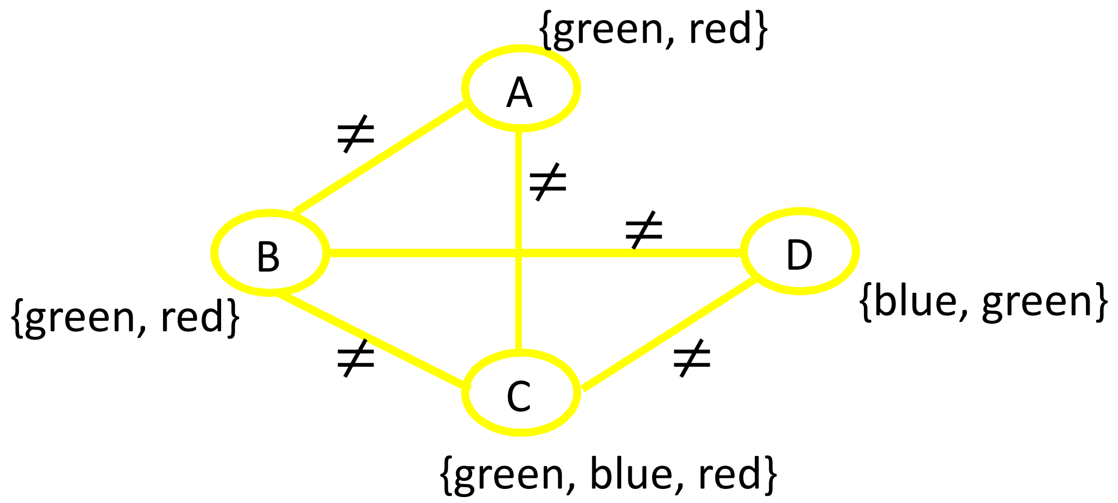
# Filtering by Forward-Checking
# Graph coloring

Assign a color to A,B,C and D (below) so that no two colors are adjacent, with possible colors for each rectangle:



A and B are green or red
C is green, blue or red
D is blue or green

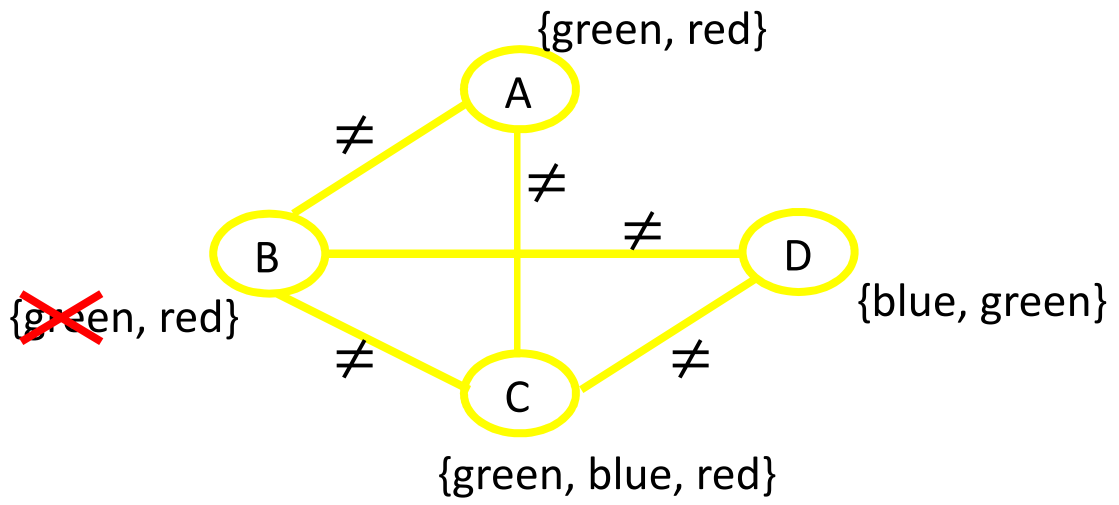
# Filtering by Forward-Checking Model



- Some pairs of variables are linked by a constraint of difference
- The domain of each variable is shown in bracket {...}.

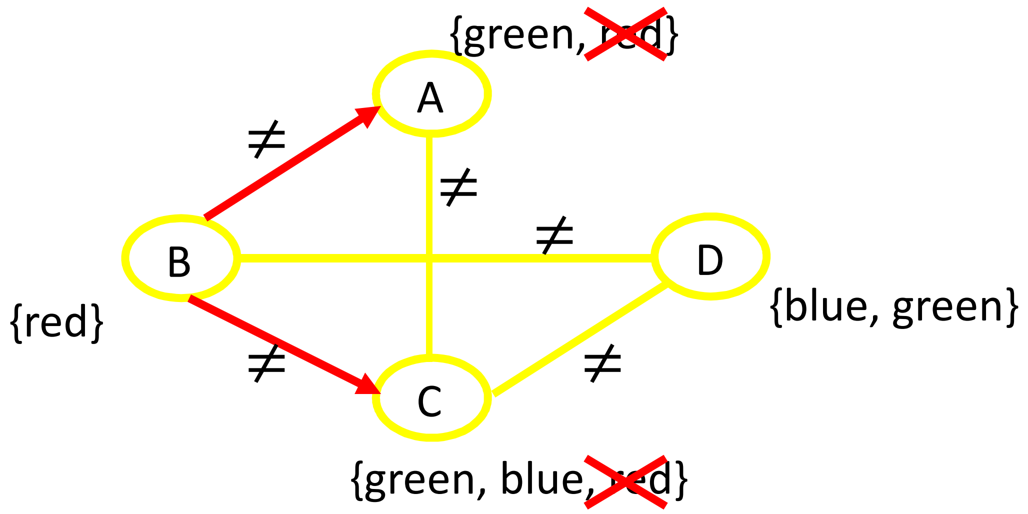
## Filtering by Forward-Checking

- Let us assume B is arbitrarily assigned to red.



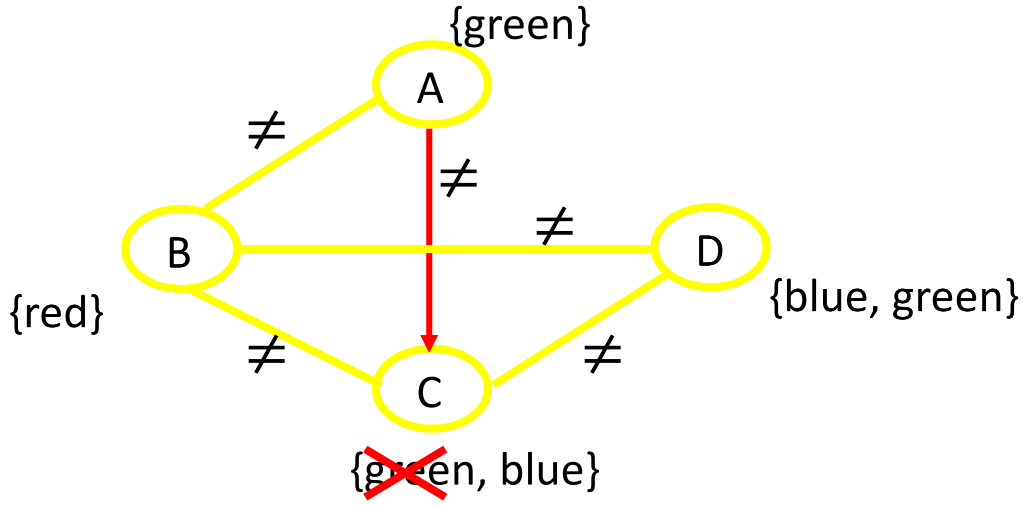
# Filtering by Forward-Checking

- Instantiation of A to green.



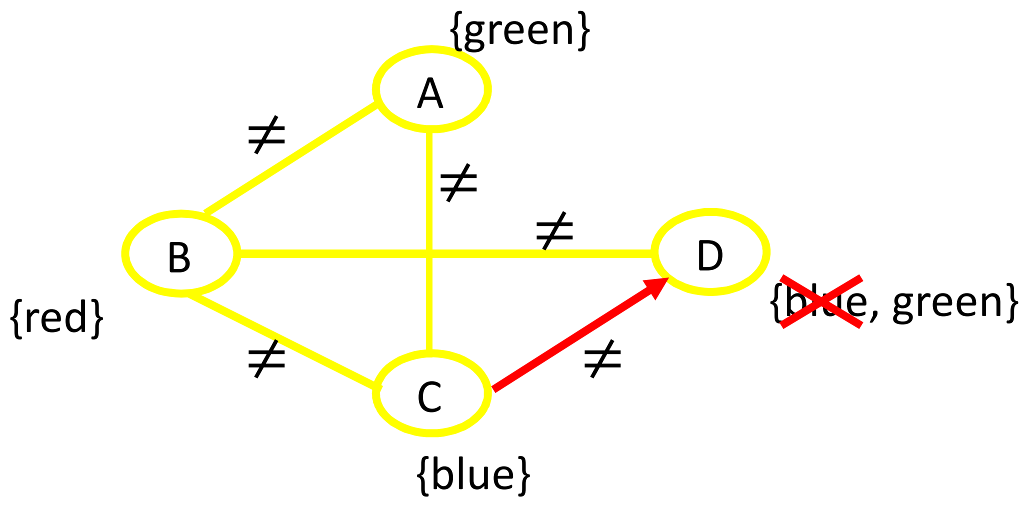
# Filtering by Forward-Checking

- Instantiation of C to blue.



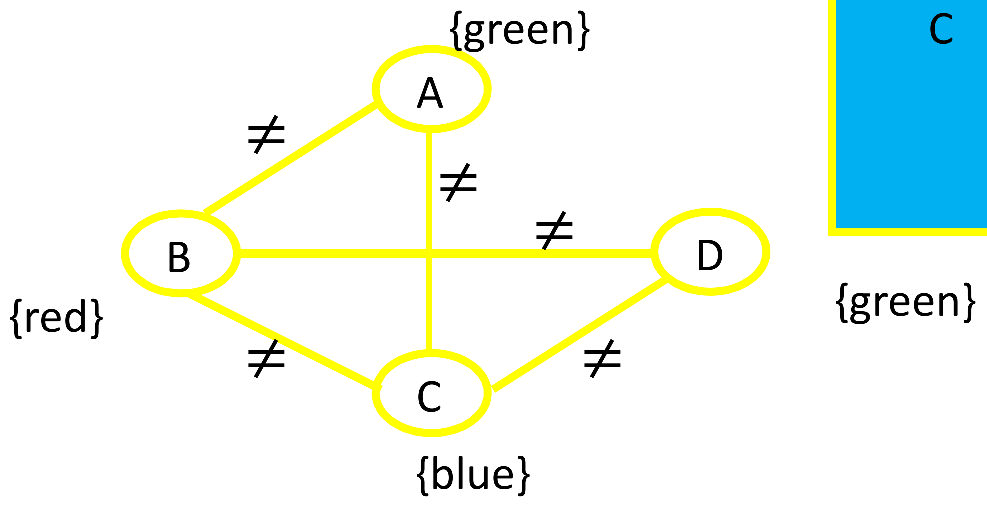
# Filtering by Forward-Checking

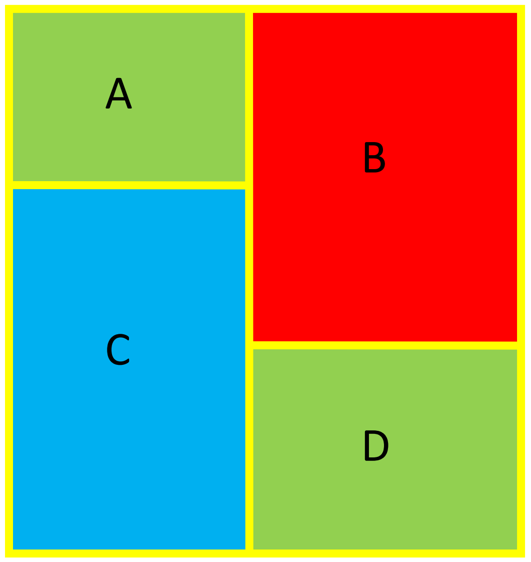
- Instantiation of D to green.



# Filtering by Forward-Checking

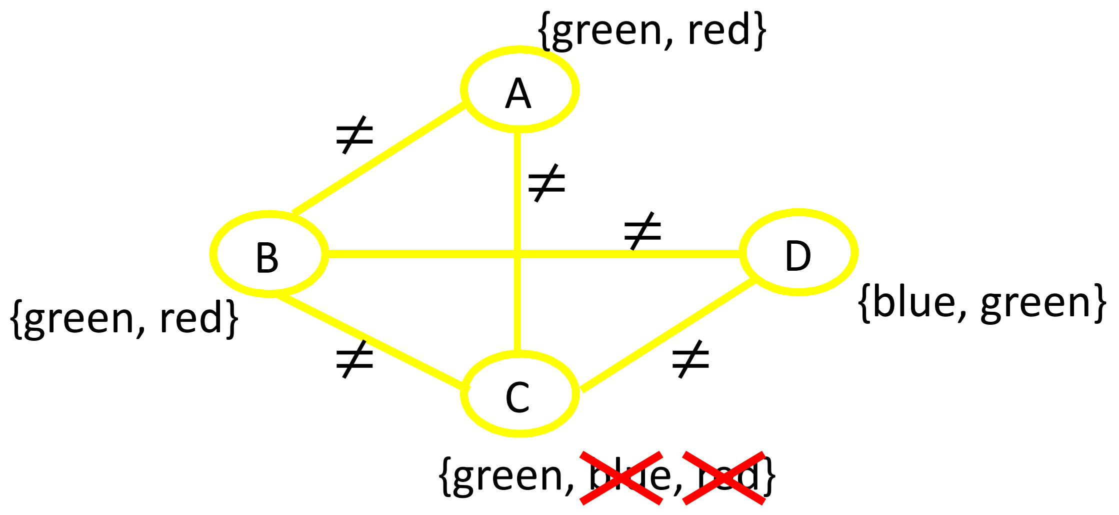
- Solution !





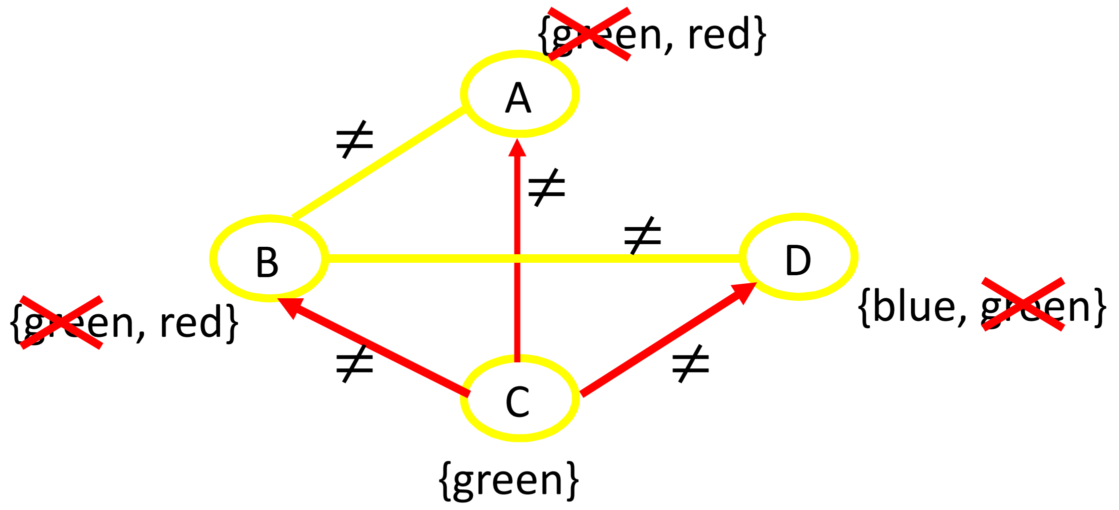
# Filtering by Forward-Checking

- Now, let us assume that C is arbitrarily assigned to green.



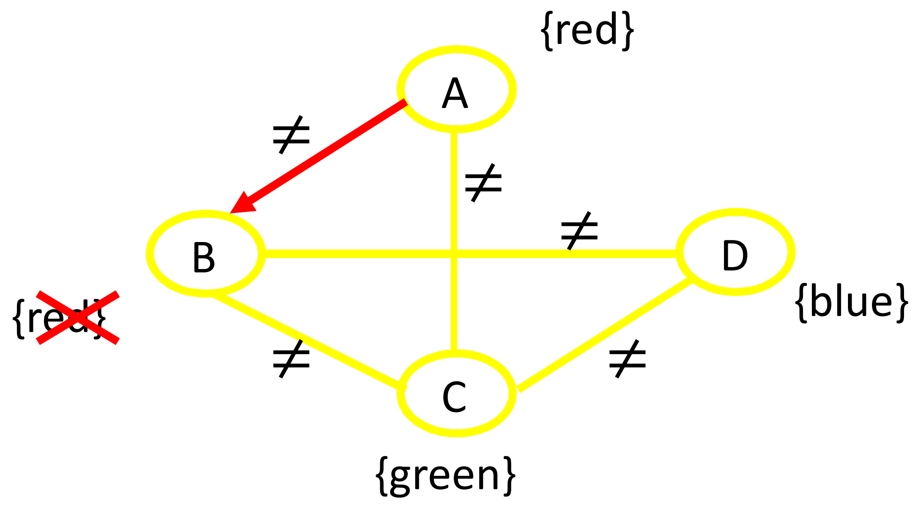
# Filtering by Forward-Checking

- Instantiation of D to blue, and A and B to red.



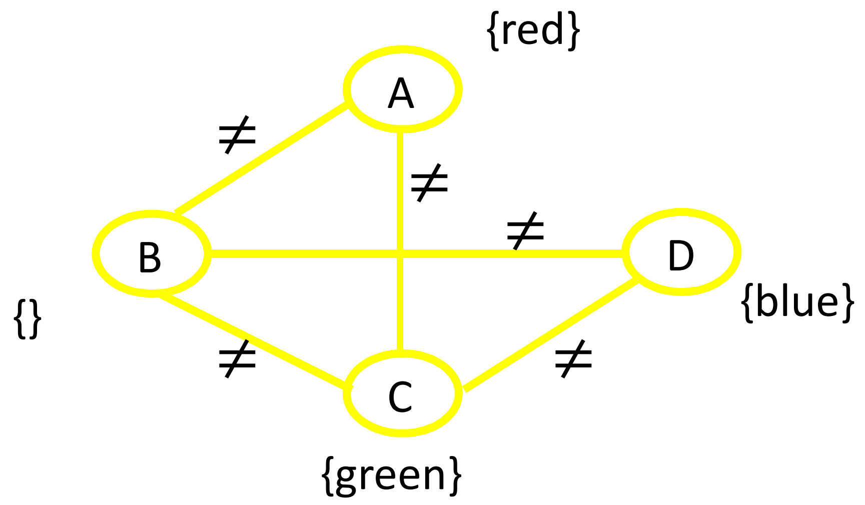
# Filtering by Forward-Checking

- Removing red from B's domain.

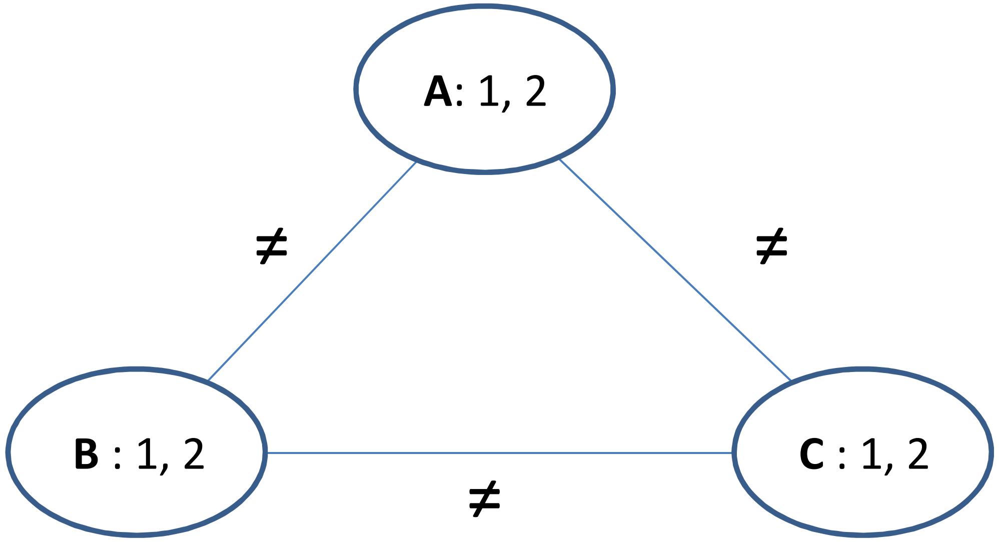


# Filtering by Forward-Checking

- B has an empty domain: failure !
- Backtrack is needed in the process…



# Arc consistancy

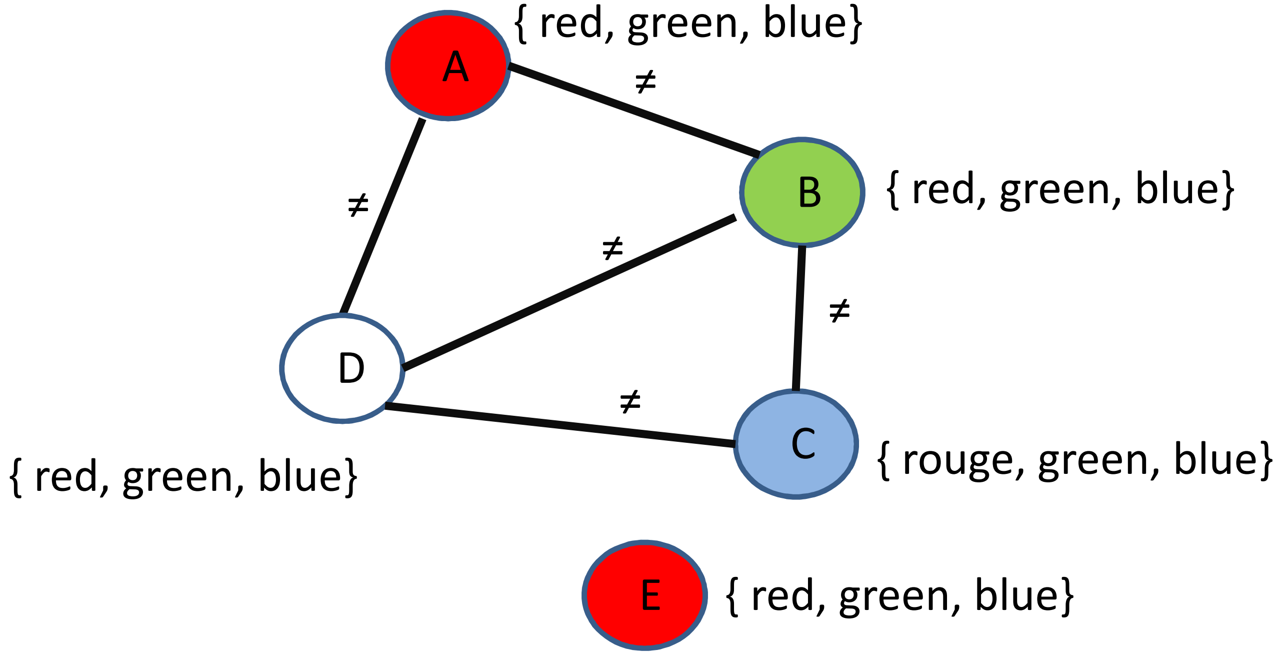


- Arc consistancy is not sufficient for the example above: path consistancy is required.
- K-consistancy, strong k-consistancy.

# Back-jumping

- The algorithm Backtrack goes back to the previous choice (e.g., lastly assigned variable).
  - Chronological backtracking.
- Back-jumping: goes back to a variable which caused the failure (e.g., the most recent) in the search tree.
- The conflict set of a variable **V** is the set of previously instantiated variables, linked to **V** by a constraint.

# Back-jumping [Russel & Norvig 2010]



- Let us assume that the order of instantiation is ***A***, ***B***, ***C***, ***E***, ***D***.
- ***A*** = red ; ***B*** = green ; ***C*** = blue ; ***E*** = red
- No value for ***D***. Chronological backtracking goes back to ***E*** !!!
- The conflict set of ***D*** is { ***A***, ***B***, ***C*** }. Back-jumping goes back to ***C*** and not ***E***.

# References

- [Russell & Norvig 2010] Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010, 3rd edition. Chapter 5.
- [Rossi et al.2006] Rossi, R.; Beek, P. V.; and Walsh, T., eds. Handbook of Constraint Programming. Elsevier. 2006.

# Conclusion

- Constraint programming is a paradigm for solving combinatorial problems.
- Difficult on real problems due to combinatorial explosion.
- Constraint programming is based on a model (variables, domains, constraints), a unique algorithm (the solver) and uses heuristics (on variables, on values).
- The algorithm uses forward checking to filter the domains of variables (removing inconsistent values). Arc consistancy is a more general algorithm for that.
- When an empty domain is detected, the algorithm goes up in the search tree (chronological backtracking, back jumping) to change a previously made choice, hoping to do better next.