An Incremental Constraint Satisfaction Algorithm for Dynamic Reconfiguration

Sina Entekhabi
Ahmet Serkan Karataş
Halit Oğuztüzün
ODTÜ, Ankara

IZTECH Dependability, 8 May 2017

Supported by TÜBİTAK-ARDEB-1001 program under project 215E188.

Outline

- Introduction
- Problem definition
- Related works
- Incremental algorithm
- Tracing example
- Conclusion
- References

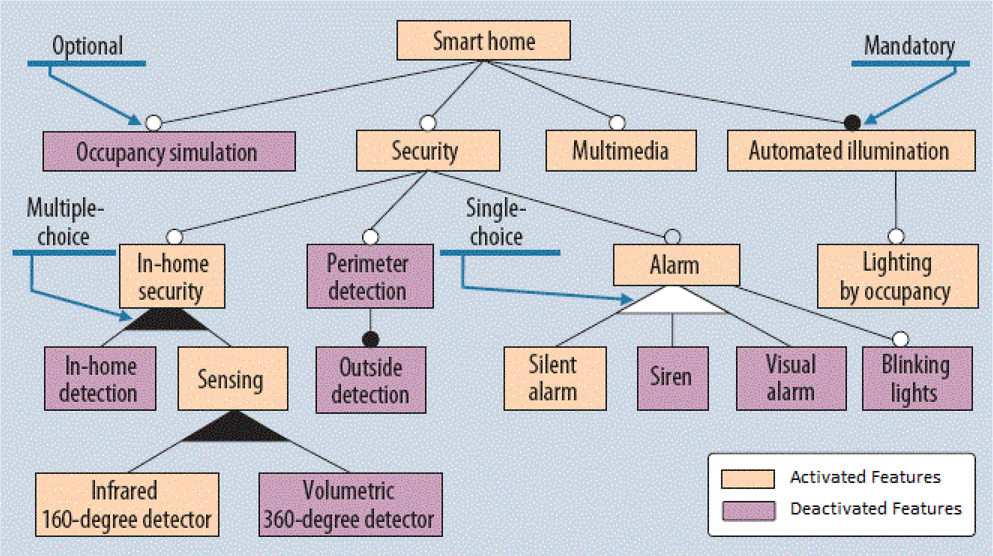
Introduction(1/4)

- Software Product Line (SPL)
  - A series of similar systems
  - Sharing common cores with some differences
  - Variability management before runtime
  - Ex: smartphones

- Dynamic SPL (DSPL)
  - Variability management at runtime
  - Ex: Smart homes

Introduction(2/4)

- Variability Management
  - Ex: Feature model (FM) diagram
  - SPL: Some of the features in a product
  - DSPL:
    - All of the features in a DSPL product
    - Runtime reconfigurations regarding context condition



Feature model diagram of a smart home[3]

Introduction(3/4)

- Constraint Logic Program
  - Containing constraints in the body of clauses
  - Ex: A(x, y):- x>0, y>1, B(x)

- FM relations can be expressed as clauses of logical expressions
  - Ex:
    - "A excludes B" as "$\neg(A \wedge B)$"
    - "A requires B" as "$A \Rightarrow B$"
    - "A is the parent of B, in a mandatory relation" as "$A \Leftrightarrow B$"
    - "A is the parent of B, in an optional relation" as "$B \Rightarrow A$"
    - "A is the parent of B and C, in an 'OR' relation" as "$B \vee C \Rightarrow A$"
    - "A is the parent of B and C, in an alternative relation" as "$((B \wedge \sim C) \vee (\sim B \wedge C)) \Leftrightarrow A$"

Introduction(4/4)

- Runtime DSPL reconfiguration

Context Monitor	
Condition 1	Resolution 1
Condition 2	Resolution 2
….	….
Condition N	Resolution N

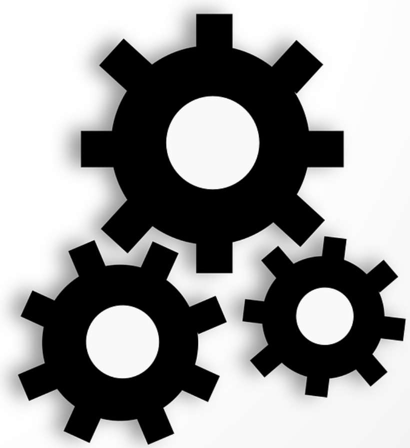
The context monitor specifies activation and/or deactivation of some of the features in specific conditions[4]

- Effective reconfiguration criteria:
  - Imposing the minimum number of changes to the current product

Problem Definition(1/2)

- The whole FM as a constraint network
    - Every relation as a constraint
    - Reaching o valid DSPL product by satisfying all of the constraints

- DSPL reconfiguration problem as Constraint Satisfaction Problem(CSP)



Problem Definition(2/2)

Having a constraint network including a set of variables V:

$$V = \{v_1, v_2, \dots, v_n\}\ where\ v_i \epsilon D_i\ for\ 1 \le i \le n,$$

and a set of satisfied constraints C among variables in V:

$$\mathrm{C} = \{c_1, c_2, \dots, c_k\},$$

and a resolution R:

$$\mathrm{R} = \{v_{j_1} \leftarrow a'_1, v_{j_2} \leftarrow a'_2, \dots, v_{j_m} \leftarrow a'_m\},$$

where the variables have the previous values:

$$v_{j_r} = a_r\ for\ 1 \le r \le m,$$

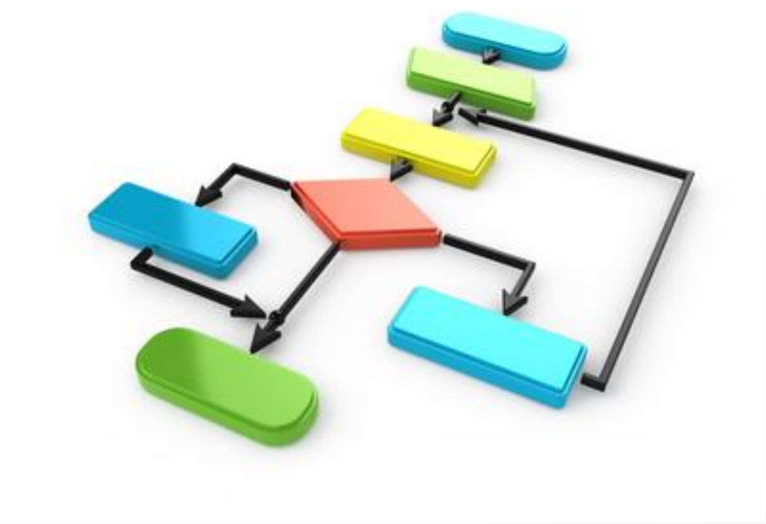
the aim is satisfying C and R while minimizing the condition θ below:

$$\theta = \sum_{r=1}^{m} a_r \oplus a'_r \text{ , where } x \oplus y = \begin{Bmatrix} 0 & if\ x = y \\ 1 & if\ x \ne y \end{Bmatrix}$$

Related works

- Incremental CSP algorithms for constraint hierarchy
  - EX: DeltaBlue, SkyBlue, cassoway

- Dynamic CSP algorithms
  - The number of constraints and/or variables are variable
  - Using previous solution or learning to reach next solution



Incremental algorithm

- Our incremental algorithm is inspired from SkyBlue
- Using the concept of multi-directional methods
- The data structure includes these parts below:
    - S-Variable
    - S-Method
    - S-Constraint
    - S-network
    - S-log
- Our algorithm includes two main functions:
    - Reconfigure function
    - Solve function

Reconfigure function

Algorithm 1 Reconfigure Function

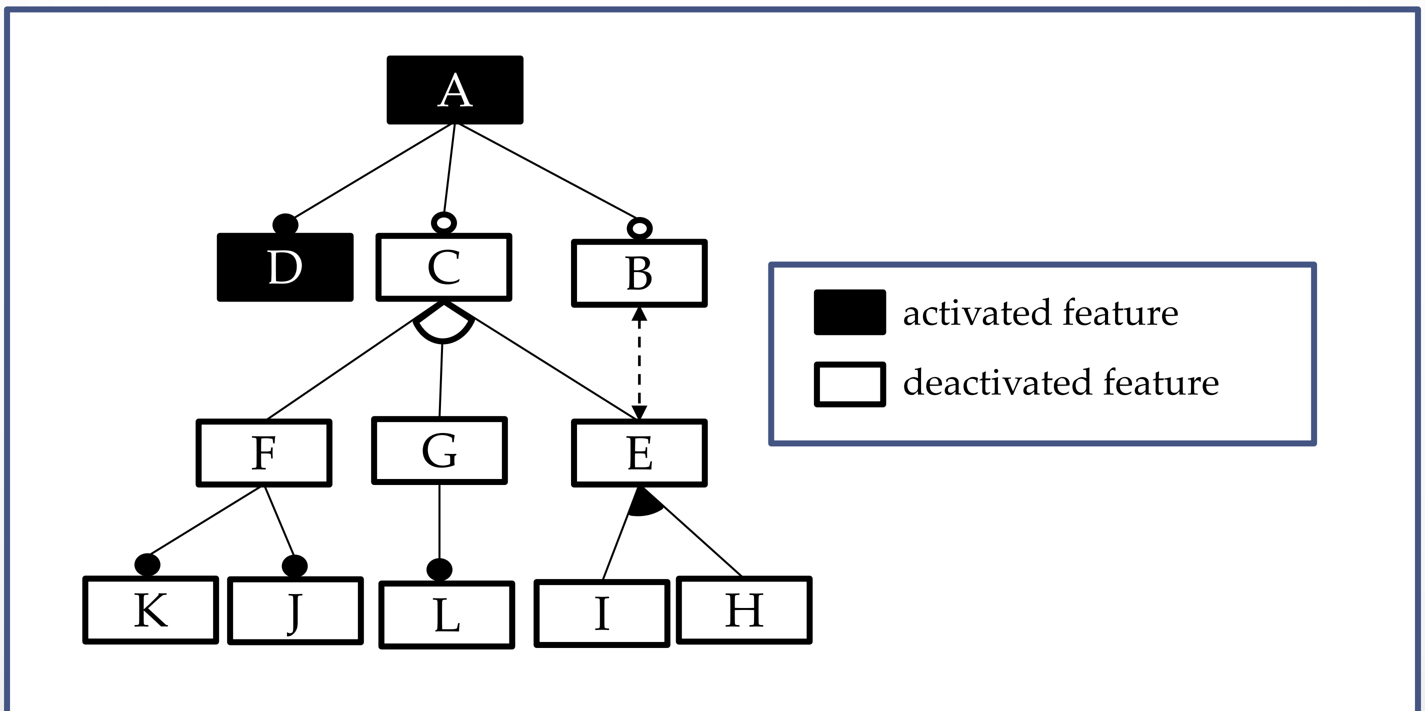
Input:
a reconfiguration request and a consistent constraint network

Output:
a list including variable changes which results in satisfying the request and a consistent constraint network or an empty list

```
function RECONFIGURE(V, System)        ▷ V - a list of variables, System - constraint network
    V_0 ← System.getVariables(getNames(V))
    newlyChangedVars ← System.setVariables(V)
    log1.assignedVariables ← V
    log1.changedVariables ← newlyChangedVars
    relatedCS ← System.relatedConstraints(newlyChangedVars)
    newCS ← sensitiveConstraints(relatedCS, newlyChangedVars)
    if newCS is empty then
        System.setVariables(V_0)
        return newlyChangedVars
    else
        result ← SOLVE(newCS, log1, System, {})
        System .setVariables(V_0)
        return result
    end if
end function
```

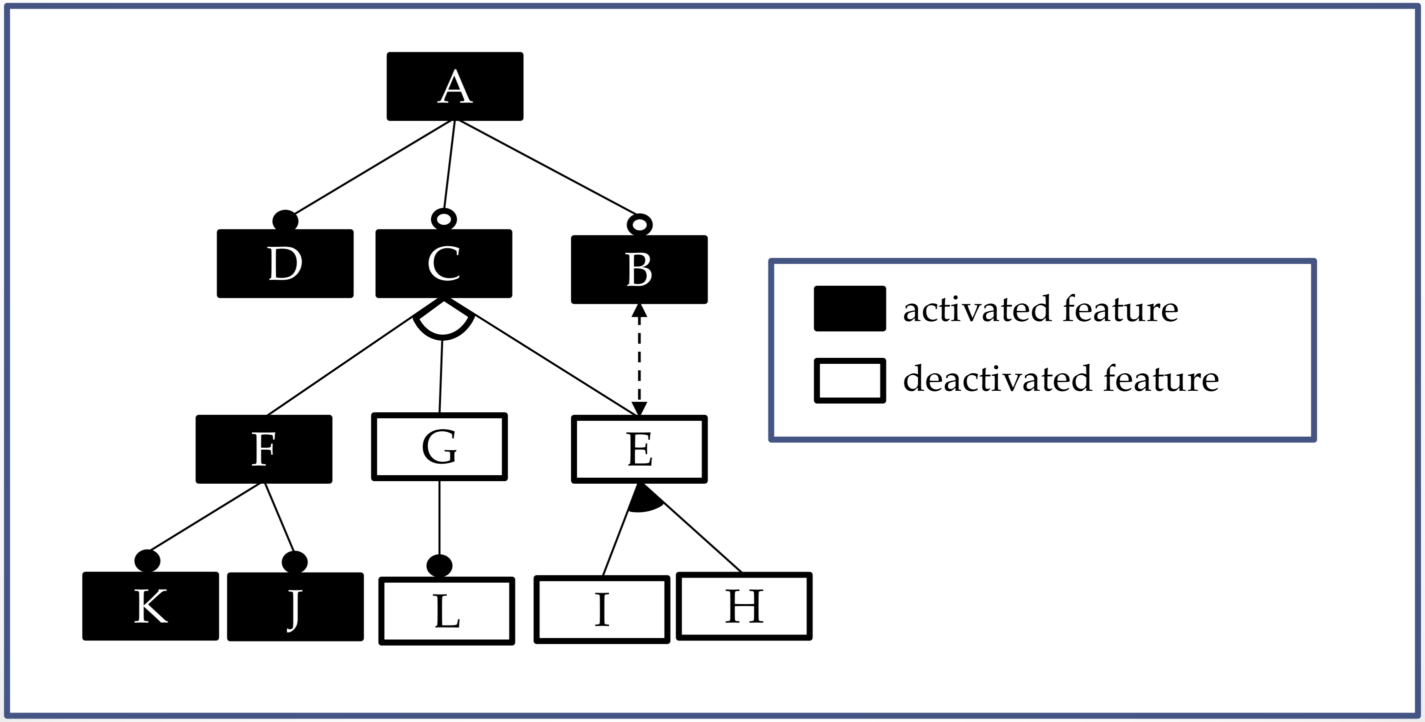
Tracing example

- Supposing a DSPL with the FM diagram below.
- Request **R**: activate Feature B and C



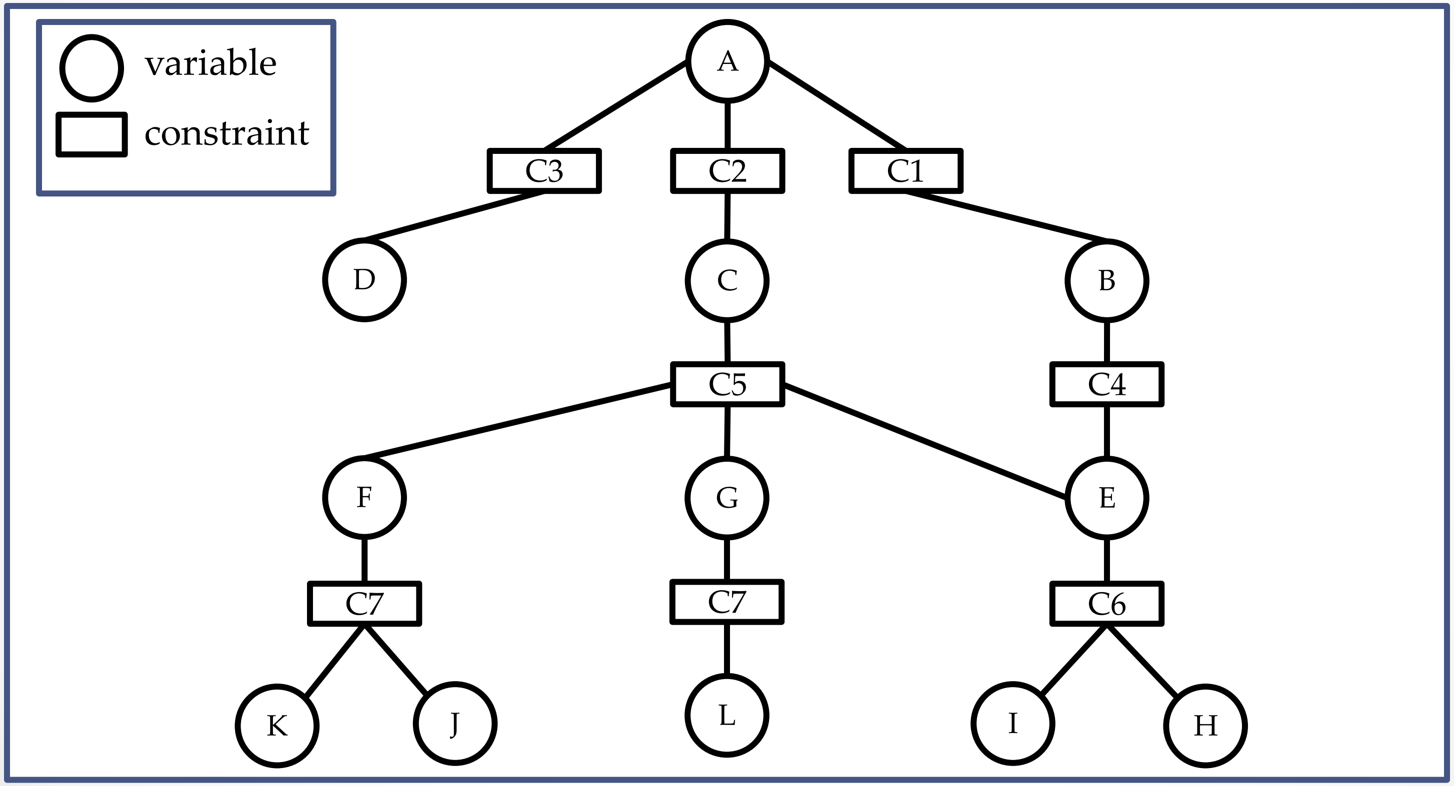
Arbitrary reconfiguration

- An arbitrary valid reconfiguration satisfying **R**
  - **changes :5**
- Valid reconfigurations with less than 5 changes exists



FM to constraint network

- Corresponding FM to a set of variables and constraints among them



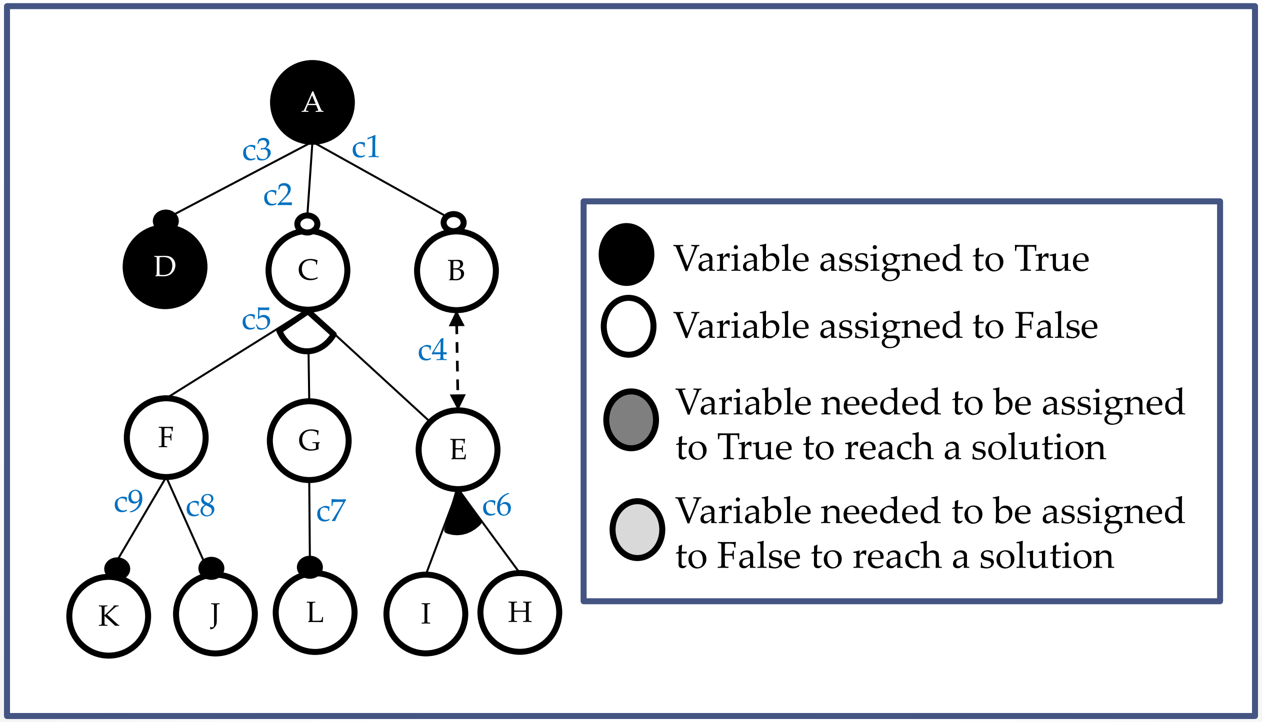
Mapping FM to a constraint network

Constraint definitions

- **C1:** $\mathrm{B} \Rightarrow A$
- **C2:** $\mathrm{C} \Rightarrow A$
- **C3:** $D \Leftrightarrow A$
- **C4:** $\sim (B \wedge E)$
- **C5:** $((\mathrm{F} \wedge \sim G \wedge \sim E) \vee (\sim \mathrm{F} \wedge G \wedge \sim E) \vee (\sim \mathrm{F} \wedge \sim G \wedge E)) \Leftrightarrow C$
- **C6:** $(\mathrm{I} \vee H) \Leftrightarrow E$
- **C7:** $\mathrm{G} \Leftrightarrow L$
- **C8:** $\mathrm{F} \Leftrightarrow J$
- **C9:** $\mathrm{F} \Leftrightarrow K$

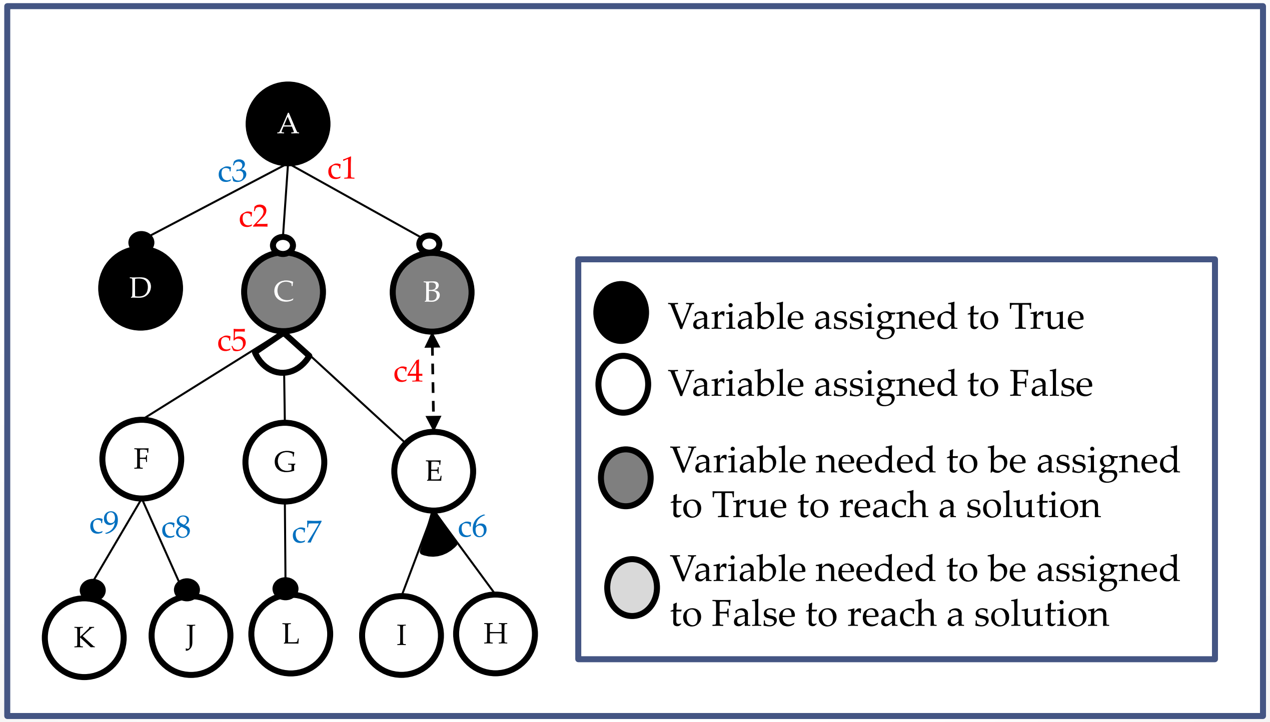
Different representation

- Representing constraint network similar to FM



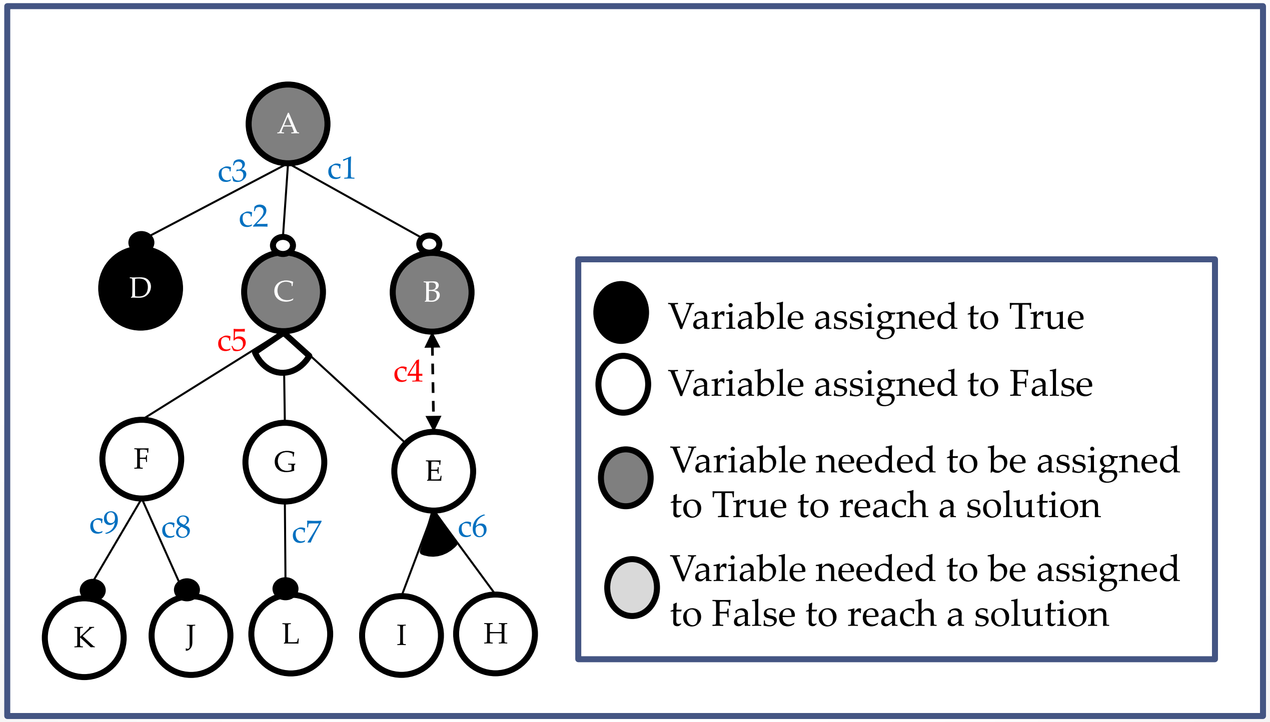
Tracing(1/11)

- Satisfying the request R as the first step
- Distributing the effects at the next steps



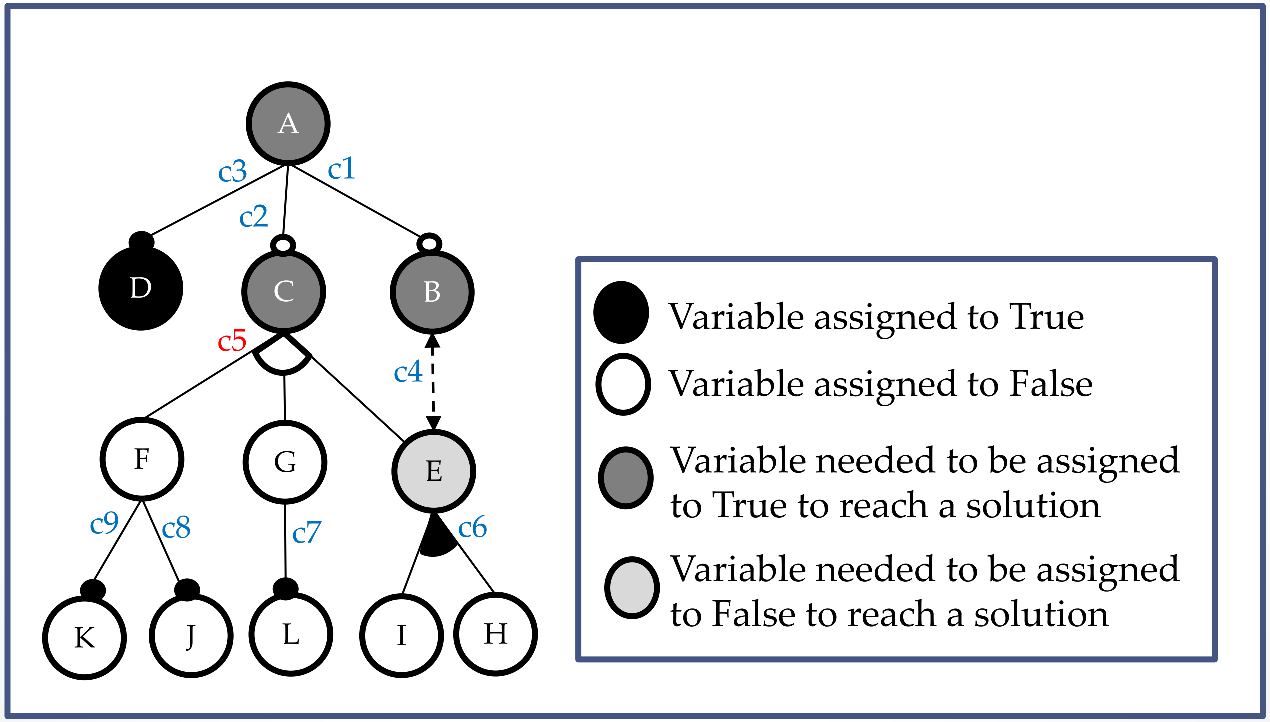
Tracing(2/11)

- A is requested to be true by C1 and C2
- A was true beforehand, no more distribution from A side



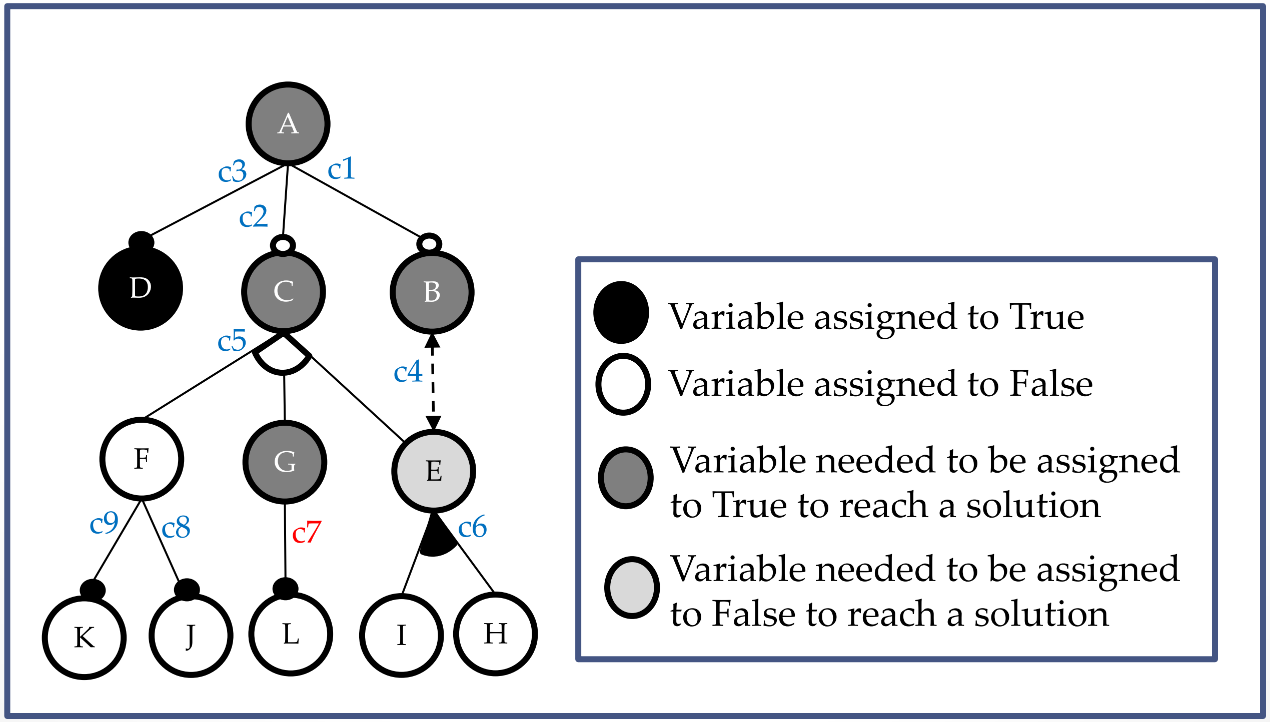
Tracing(3/11)

- E is requested to be False by C4



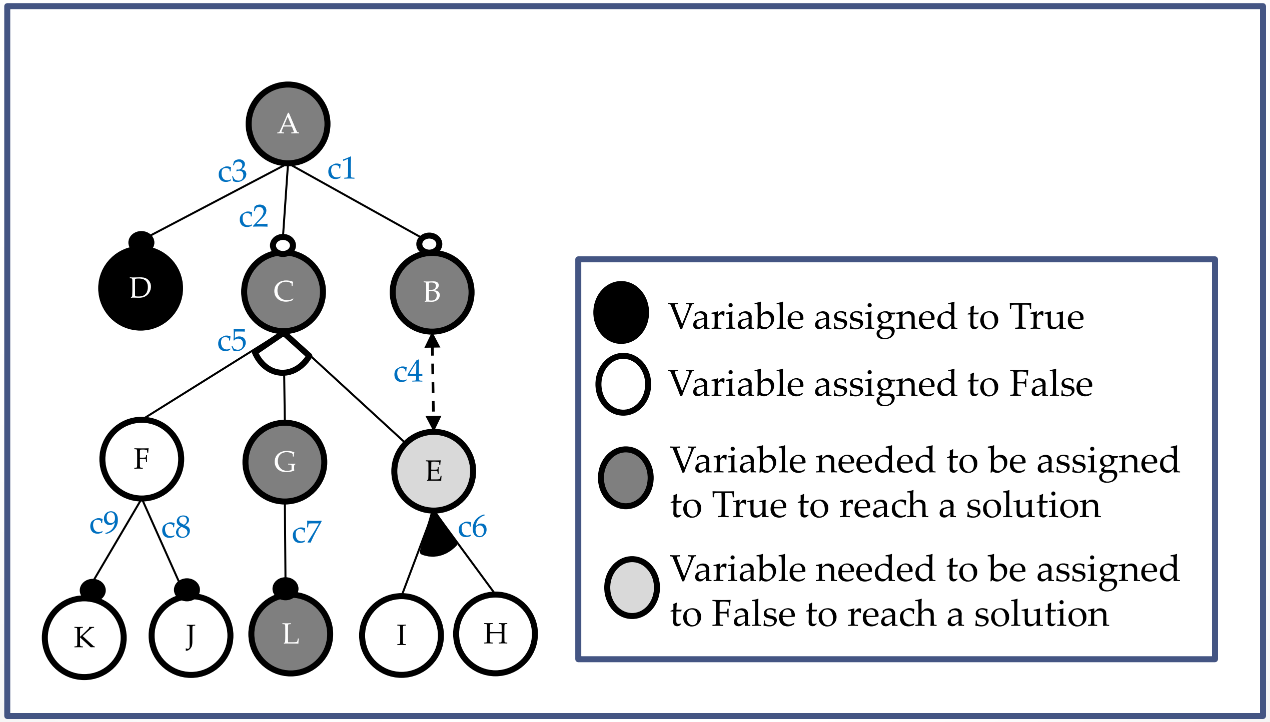
Tracing(4/11)

- C5 needs G or F be True, but not E
- Choosing G arbitrarily at this point



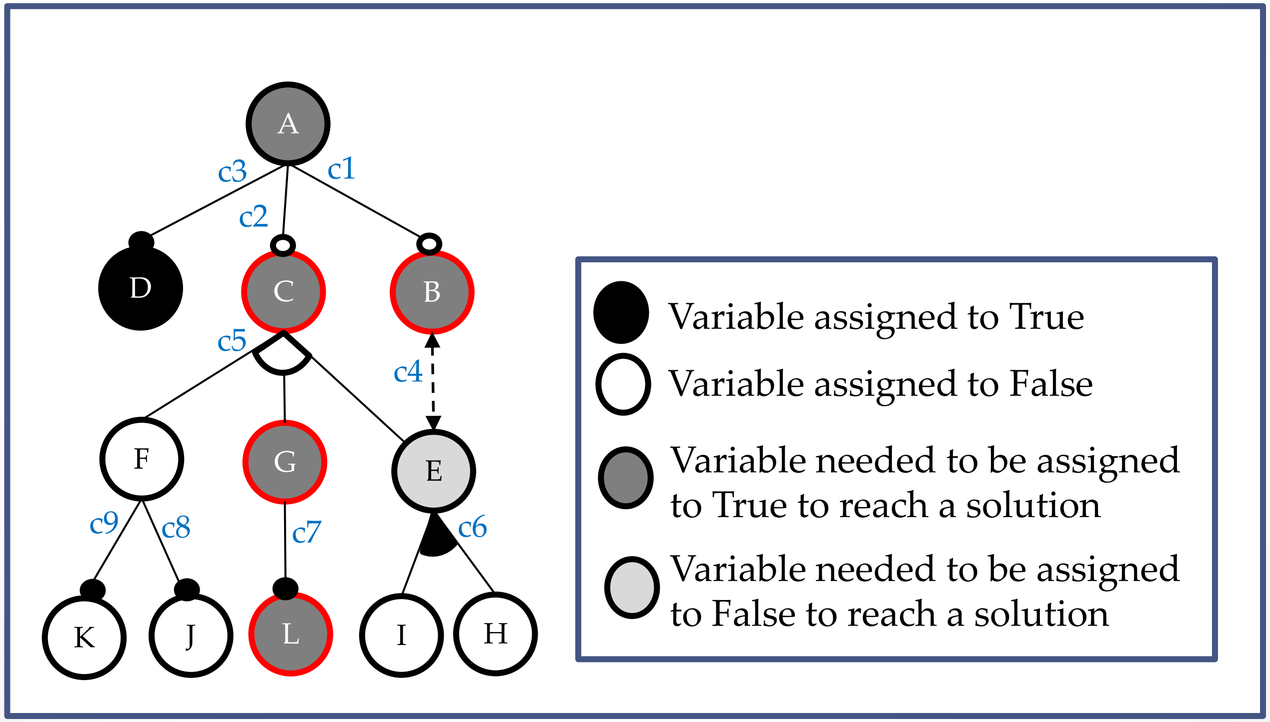
Tracing(5/11)

- C7 needs L be True.



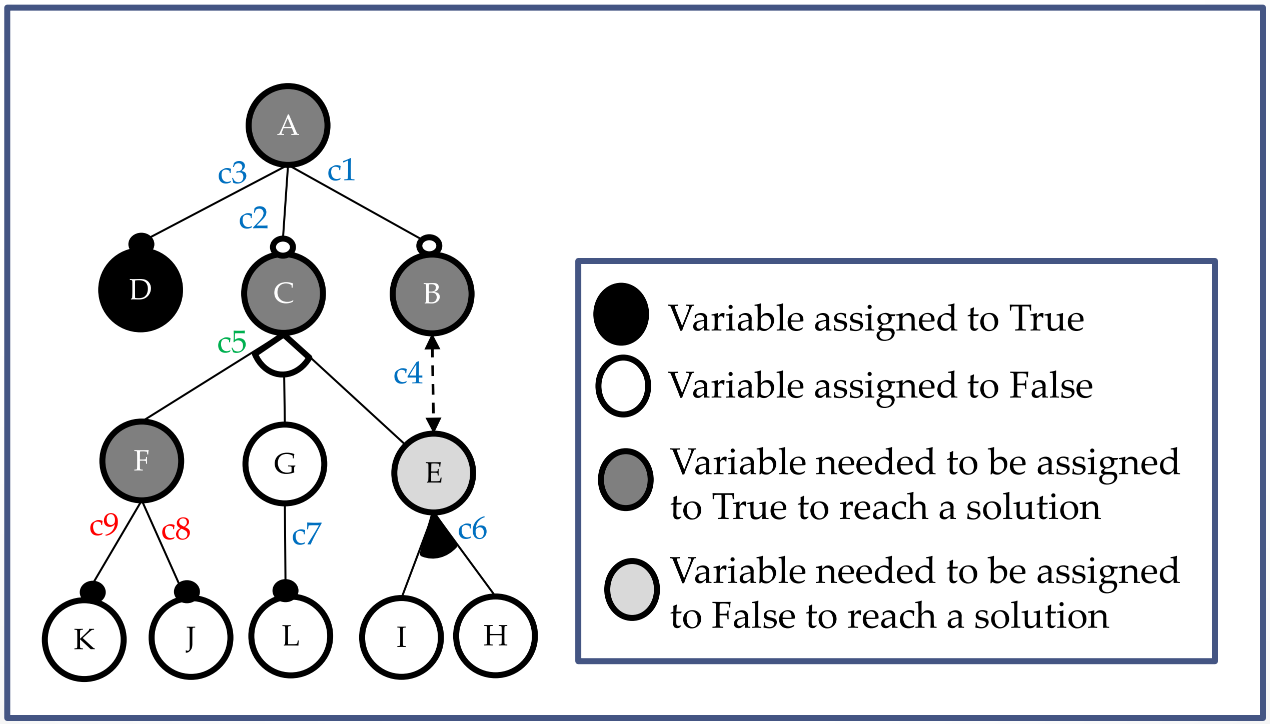
Tracing(6/11)

- No more solution to recheck: one Solution found
- Solution 1: change (B,C,G,L) to true, changes:**4**



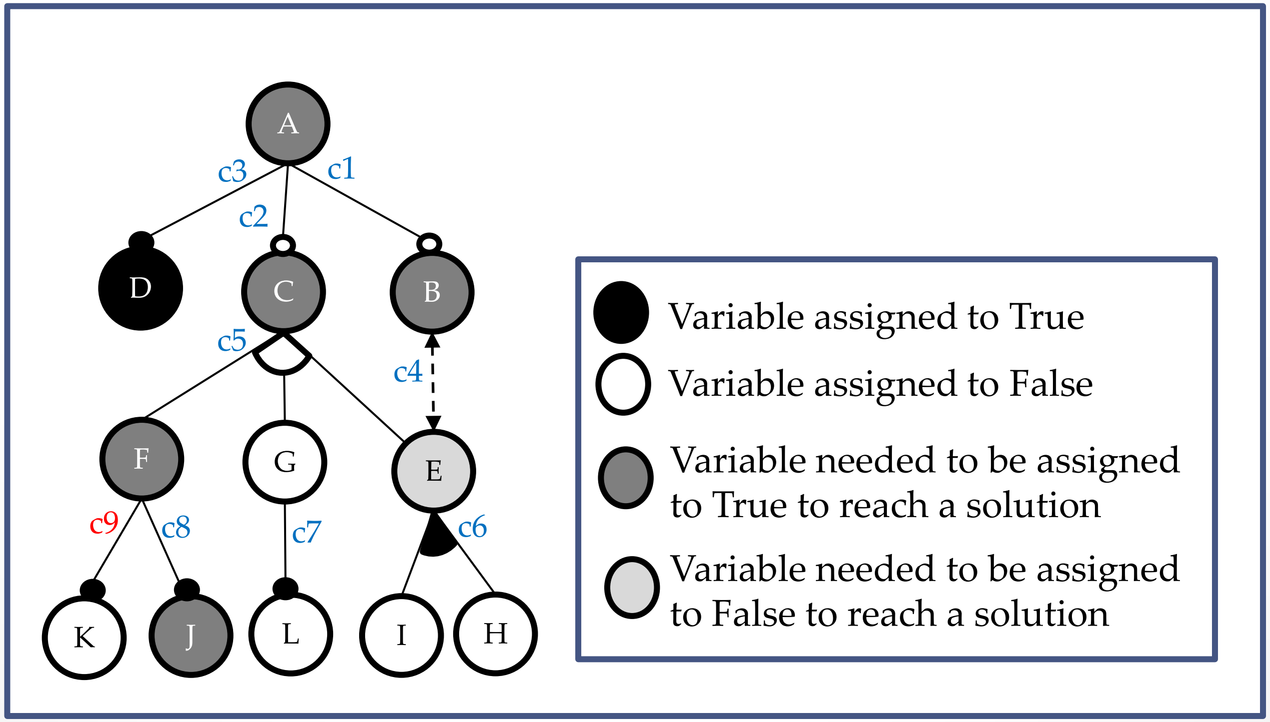
Tracing(7/11)

- Backtrack: to satisfy C5, F can be True as well.
- Choosing F and trying to find a solution



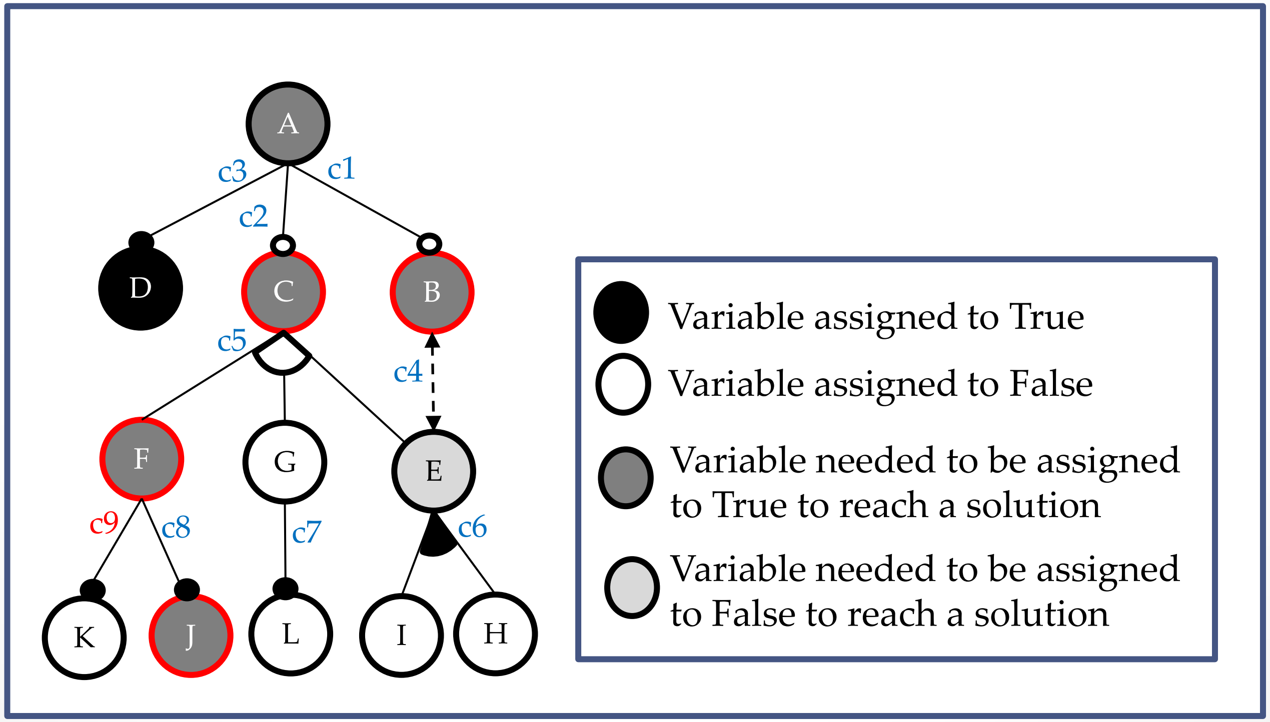
Tracing(8/11)

- C8 needs J be True.
- Having 4 changes up to now in this solution search.



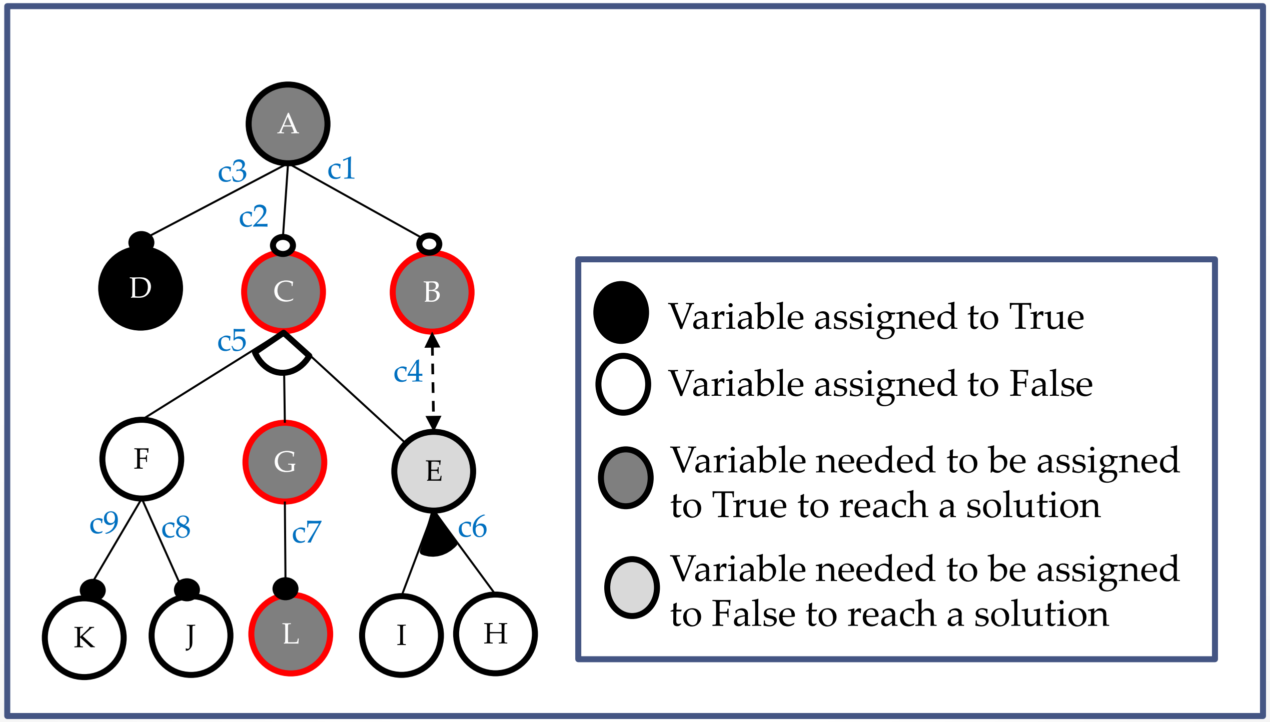
Tracing(9/11)

- Having 4 changes up to now in this solution search.
- Solution1 had **4** changes as well. Pruning this branch



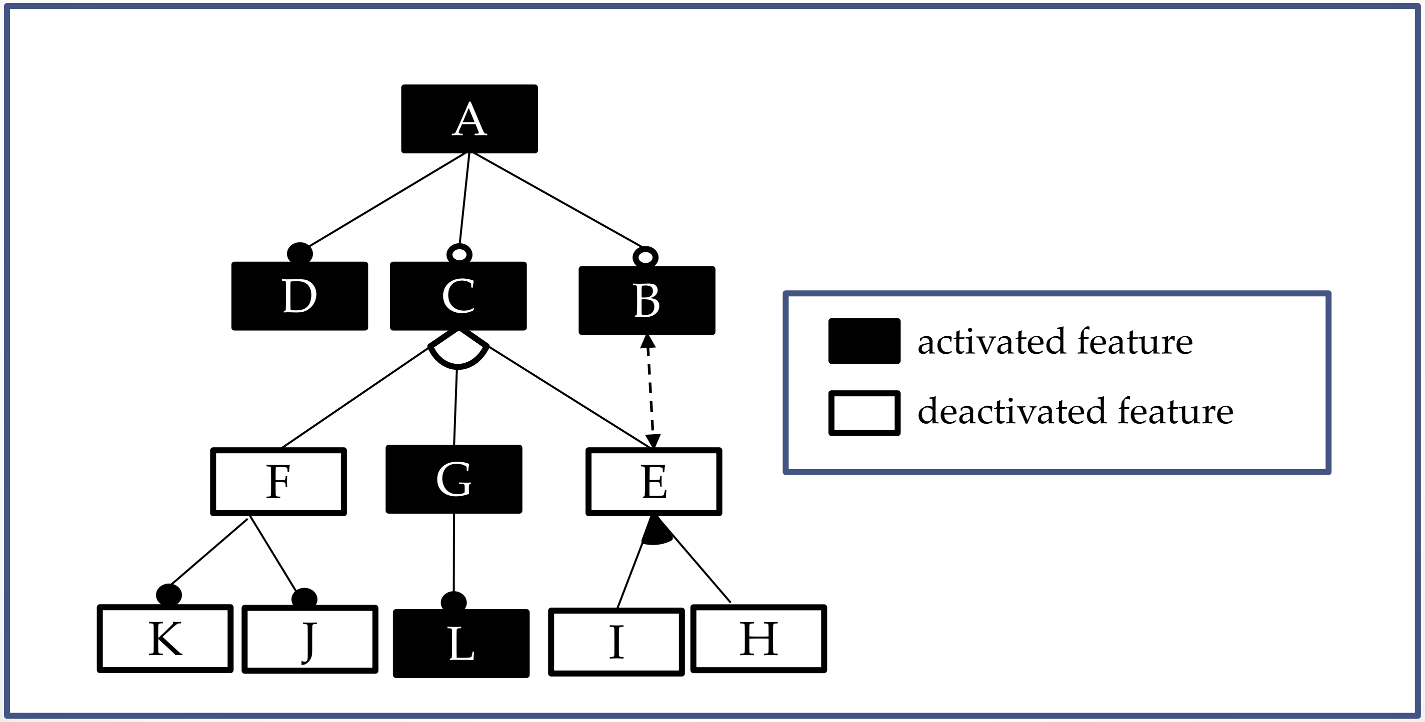
Tracing(10/11)

- The algorithm return an optimum solution, solution1.
- Solution1 is the only optimum solution in this case.



Tracing(11/11)

- After applying solution1 to the system, the FM of the system would be the diagram below.



Conclusion

- Variability management of DSPLs by FM
- FM corresponds to constraint logic program
- Dynamic reconfiguration in DSPLs as CSP
- Effective reconfiguration by incremental algorithms

References(1/2)

- [1] P. Clements and L. Northrop, Software product lines. Addison-Wesley,2002.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," DTIC Document, Tech. Rep., 1990.
- [3] N. Bencomo, P. Sawyer, G. S. Blair, and P. Grace, "Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems." in SPLC (2), 2008, pp. 23–32.
- [4] Pau Giner, Joan Fons, Vicente Pelechano, Carlos Cetina, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes", Computer, vol. 42, no. , pp. 37-43, October 2009, doi:10.1109/MC.2009.309
- [5] CETINA, Carlos, et al. Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer, 2009, 42.10.

References(2/2)

- [6] D. Benavides, P. Trinidad, and A. Ruiz-Cort´es, "Automated reasoning on feature models," in Seminal Contributions to Information Systems Engineering. Springer, 2013, pp. 361–373.
- [7] A. S. Karataş, H. Oğuztüzün, and A. Doğru, "From extended feature models to constraint logic programming," Science of Computer Programming, vol. 78, no. 12, pp. 2295–2312, 2013.
- [8] B. N. Freeman-Benson, J. Maloney, and A. Borning, "An incremental constraint solver," Communications of the ACM, vol. 33, no. 1, pp. 54– 63, 1990.
- [9] M. Sannella, "Skyblue: a multi-way local propagation constraint solver for user interface construction," in Proceedings of the 7th annual ACM symposium on User interface software and technology. ACM, 1994, pp. 137–146.
- [10] G. J. Badros, A. Borning, and P. J. Stuckey, "The cassowary linear arithmetic constraint solving algorithm," ACM Transactions on Computer- Human Interaction (TOCHI), vol. 8, no. 4, pp. 267–306, 2001.


THANK
YOU !