Rule-Based Visualization of Tableau Calculus for Propositional Logic

Nada Sharaf, Slim Abdennadher
Computer Science and Engineering Department
The German University in Cairo
Cairo, Egypt
{nada.hamed, slim.abdennadher}@guc.edu.eg

Thom Frühwirth
Institute of Software Engineering and Compiler Construction
University of Ulm
Ulm, Germany
thom.fruehwirth@uni-ulm.de

***Abstract*—This paper discusses a rule-based approach for visualizing proofs using tableaux techniques. Semantic Tableau is usually used to prove by refutation. In addition, tableaux techniques are commonly studied in different courses. Visualization is an effective teaching methodology. The availability of visualization methods that could be used by instructors is thus important.**

***Index Terms*—Tableau Proof, Visualization, Constraint Handling Rules**

I. Introduction

Semantic tableaux is a procedure used for checking satisfiability [1] in propositional logic. Tableaux basically tries to search for a model. If such a satisfying interpretation was found then the expression is marked as satisfiable. Semantic Tableaux could be applied for first-order logic formulas as well as propositional logic. In the paper, we focus on propositional logic for the proof of the concept. In this case, formulas could be checked for satisfiability. In case the formula is satisfiable, the tableaux would be able to find a model for it. For example a formula $a \wedge b \wedge (\neg a \vee \neg b)$ is unsatisfiable. In this case, all the branches of the corresponding tableau will be closed branches indicating the lack of a model.

This methodology has been used in different contexts and logics [2] [3]. Due to the importance of this proof technique, it is also studied in different curricula.

The existence of visualization methodologies have proven to be effective in teaching [4]. Throughout different studies, groups using algorithm visualisation techniques had better results than groups with classical methodologies [5]. Such visualization platforms could additionally be used by instructors in classes and office hours.

The aim of the presented work is to provide a tool that students and instructors would benefit from in the context of tableaux proof techniques. With the new tool, the student could evaluate any expression. The tool would thus be helpful in learning as students get to see in a step-by-step manner how the computation is done. In other words, the tool incorporates the strength of visualization into studying tableaux proof techniques. Instructors can also use visualization to provide easy-to-follow examples in the classrooms. The tool uses rules to represent the tableaux proof technique. In addition, the visuaization details are represented using the so-called annotation rules [6]–[8]. This makes the engine of the tool declarative and extensible.

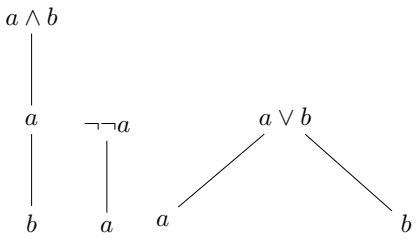
The paper is organized as follows: Section II introduces semantic tableaux in propositional logic using examples. Section III introduces Constraint Handling Rules (CHR). Section IV shows how the Tableaux technique was implemented using rules through the use of CHR. Section V shows how the tableau proof technique was visualized. The section gives some examples as well. The paper finally concludes with directions to future work.

II. Semantic Tableaux

Semantic Tableaux tries to build back the truth table. Semantic tableaux is in many cases used to prove by refutation. In other words, in case the statement p is to be checked, we would try to prove that $\neg p$ is false. That way, p must be true [1], [9].

With propositional logic however, we can use semantic tableau to prove the satisfiability of an expression.

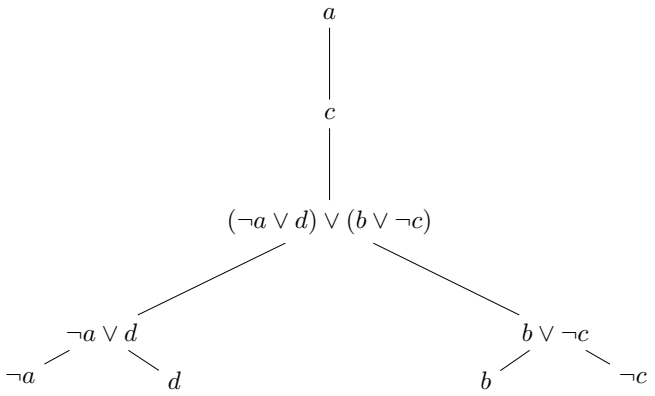
Without loss of generality, having rules for $\wedge, \vee$ and $\neg$ would produce sound and complete results. The rules are given below:



An expression $a \wedge b$ means that a and b have to be true. On the other hand, the expression $a \vee b$ means that if a or if b is true, the expression holds. That is why branching occurs since it could be that one of the branches fails but the expression still holds. Finally, $\neg\neg a$ means that a has to be true. These rules are able to handle different types of expressions. A branch that contains a contradiction is a closed branch and can never lead to a solution.

For example the expression: $a \wedge c \wedge ((\neg a \vee d) \vee (b \vee \neg c))$ produces the tableau:

978-1-5386-7202-0/18/$31.00 ©2018 IEEE
DOI 10.1109/iV.2018.00069



In this case, the first branch is a closed one due to the existence of a and $\neg a$ in it. The last branch is also closed because it contains c and $\neg c$. The other two open branches represent two possible models to the expression. The first model is having a, c and d as true statements. The second model is to have a, c and b as true statements.

III. Constraint Handling Rules

This section introduces Constraint Handling Rules (CHR) [10]–[12]. CHR is a declarative rule-based language. CHR was introduced for writing constraint solvers. However, due to its declarativity and ease-of-use, it has developed into a general purpose language. The rules of a CHR program act on constraints in the constraint store. There are two types of constraints: CHR or user-defined constraints and built-in constraints handled by the host language. A CHR rule consists of a head (H), a body (B) and an optional guard (G). The guard is a condition for applying the rule. A rule could also have a name. A head contains CHR constraints only. A guard could only have built-in constraints. The body, on the other side, could contain CHR or built-in constraints. A CHR program consists of a set of "simpagation" rules. A CHR rule is applicable if the store contains constraints matching the constraints in the head of the rule and if the guard of the rule is satisfied [11]. A simpagation rule has the form:

$$optional\_rule\_name \ @ \ H_K \setminus H_R \ \Leftrightarrow \ G \mid B.$$

A simpagation rule has two types of head constraints: H_K and H_R. H_K are the kept head constraints while H_R are the removed ones. On applying a simpagation rule, constraints matching H_R are removed from the constraint store while the ones matching H_K are kept in the store. There are two special types of simpagation rules: simplification and propagation rules. A simplification rule is a simpagation rule with empty H_K while a propagation rule has an empty H_R. A simplification rule has the form:

$$optional\_rule\_name \ @ \ H_R \ \Leftrightarrow \ G \mid B.$$

Simplification rules replace the head constraints (H_R) with the body constraints (B). A propagation rule has the form:

$$optional\_rule\_name \ @ \ H_K \ \Rightarrow \ G \mid B.$$

If a propagation rule is applied, it adds the body constraints (B) to the constraint store. No constraints are thus removed from the store [11]. A wide range of algorithms were implemented through CHR. The following single-ruled program is able to find the smallest number among a set of numbers. The numbers are represented by the constraint `min/1`. Every time the rule `find_min` is applied, two numbers (`A` and `B`) are compared against each other. The guard makes sure that `A` is less than `B`. The smaller number is kept in the constraint store and the larger number is removed from the store. On successive application of this rule, only the smallest number survives.

```
find_min @ min(A) \ min(B) <=> A<B | true.
```

The following example shows the constraint store after each application of the rule for the query `min(6), min(5), min(4)`. The underlined constraints represent the ones that matched the head of the rule. The first application of the rule removes `6` and keeps `5`. The second application removes `5` and keeps only the constraint holding `4` in the constraint store. `4` is indeed the minimum number.

$$\underline{min(6), min(5)}, min(4)$$
$$\downarrow$$
$$\underline{min(5), min(4)}$$
$$\downarrow$$
$$min(4)$$

IV. Tableaux with Constraint Handling Rules

CHR could be used to implement Boolean constraint systems [13], [14]. To implement the Tableaux proof technique with CHR, different constraints were used. Each branch is labeled with an identifier. Once a branch `B` is added, it is assumed to be a successful branch by automatically adding the constraint `success(B)`. Each component in a branch is represented by the constraint `branch/3`. For example, the constraint, `branch(BNum,L,E)`, represents the existence of expression `E` in branch `BNum` at level `L`. In addition, the total number of branches is represented by the constraint `branchesT(Num)`. The constraint `branch/2` represents the number of nodes in a branch. `branch(2,1)` means that the branch identified by `2` has one node only.

A branch fails once it contains an expression and its negation. This is represented by the CHR rule:

```
branchF @ branch(BNum, Level1 ,A), branch(BNum, Level2 ,
    not(A)) ==>bFail(BNum).
```

In case a branch has a disjunction, branching happens. The rule `oring`, handles this case. The rule is fired once branch `BNum` has a disjunction at level (`Level`). In this case, two new branches are added. The first branch is identified by the number `NumB`. The second branch is identified with `NumB1`. Using the constraint `bAnces/2`, the old branch is marked as an ancestor for the two new branches. The rule is only executed if the disjunction occurs at a so-far successful branch. Otherwise there is no need to continue processing the branch.

```
oring @ success(BNum), branch(BNum,Max)\branch(BNum,
    Level,or(A,B)) , branchesT(NumB) <=> NumB1 is
    NumB + 1, NumB2 is NumB + 2 | branch(NumB,1),
    success(NumB), branch(NumB,0,A) bAnces(NumB,BNum)
```

```
, branch(NumB1,1),success(NumB1), branch(NumB1
,0,B), bAnces(NumB1,BNum), branchesT(NumB2).
```

The rule `failSucc` sets a branch to fail if one of its ancestors is a failed branch.

```
failSucc @ bFail(BNum), bAnces(NewBranch,BNum) ==>
    bFail(NewBranch).
```

The rules `branchF2`, and `branchF3`, handle the cases where a branch and its ancestor have contradicting nodes in this case the child branch fails.

```
branchF2 @ branch(BNum1,Level1,A),branch(BNum2,
    Level2,not(A)),bAnces(BNum,BNum1) ==>bFail(BNum)
    .

branchF3 @ branch(BNum1,Level1,not(A)),branch(BNum2,
    Level2,A),bAnces(BNum,BNum1) ==>bFail(BNum).
```

The rule `newAnc` handles the transitivity of the ancestor relation. It states that whenever `B` is an ancestor of `A` and `C` is an ancestor of `B` then `C` is an ancestor for `A`.

```
newAnc @ bAnces(A,B), bAnces(B,C) ==> bAnces(A,C).
```

In case the unsatisfiability is to be checked for, the following rule is added:

```
cleanUp @ finish \ success(_)<=>fail.
```

The rule `cleanUp` makes sure the program fails if there is a branch that succeeds.

An example of querying the program is

```
start,expression(and([a,not(a),or(b,or(c,d))])),
    finish.
```

In this case the expression to be checked is $a \wedge \neg(a) \wedge (b \vee (c \vee d))$. As seen from the query the expression is embedded within the constraint `expression`. Two auxiliary constraints are also added: `start` and `finish`. The constraint `start/0` initializes the procedure. It triggers the rule:

```
ruleStart @ start <=> branchesT(1),branch(0,0),
    success(0).
```

The rule `ruleStart` adds the initial branch identified with `0`. The total number of branches are thus one. The total number of nodes in branch `0` is set to be zero. The rule `expr` handles the initial expression entered through the query. The rule `expr` adds the conjuncts of the expression to the main branch (branch 0).

```
expr @ branch(0,Num),expression(and([A|B])) <=> Num1
     is Num + 1 | branch(0,Num1), branch(0,Num,A),
    expression(and(B)).
```

V. Visualization of Tableaux Proofs

The idea is to use the graph visualization methodology introduced in [15] in addition to the animation procedure introduced in [6] to animate the CHR implementation of the proof technique.

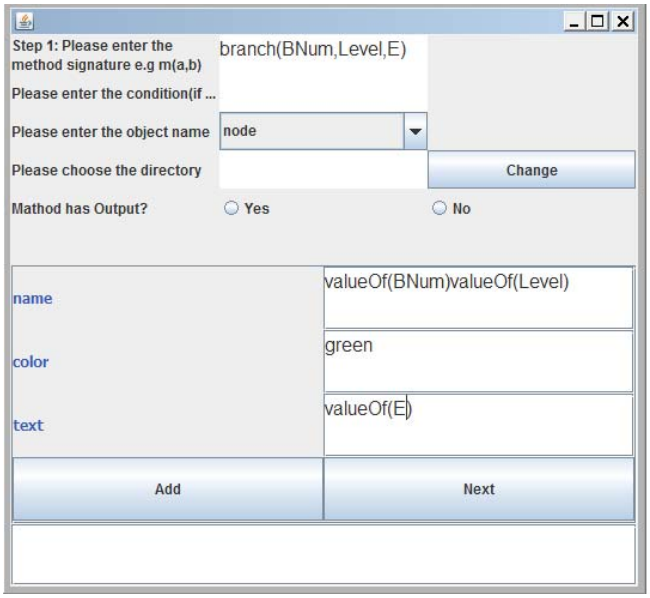


Fig. 1. Annotating a branch

A. Annotation Rules for Animating CHR Programs

In [6], annotation rules were used in *CHRAnim* to visualize the execution of CHR programs. The idea was to associate an occurrence of CHR constraint with a visual object. Thus, whenever the constraint was added/removed, the corresponding visual object(s) gets added/removed. By time, this would produce an animation of the algorithm implemented by the program. The annotated constraints in this case are the "interesting constraints" whose occurrence affect the data structure manipulated by the program. Users are provided with an interface through which they can associate a constraint with a graphical object as shown in Figure 1. Once the user chooses the graphical object, the panel is populated with the corresponding parameters. *CHRAnim* could be used with any visualization platform. The engine is able to read a file with the available objects. *CHRAnim* outsources the actual visualization to an existing platform to move the intelligence of the system to the annotations instead. Each parameter can have a different type of value including:

1) A constant value e.g. green, 100.
2) The result of the function `valueOf(Arg)` such that `Arg` is one of the arguments of the annotated constraint.

The value could also be an expression hat contains combinations of constant values and outputs of the `valueOf` function. The annotation could also have a precondition for application.

In addition, a rule could be annotated to have a visual effect [6]. In general the idea, is that the rule is annotated with a constraint that is then given normal graphical annotations. This is also done through the interface. In this case, once the rule is executed, a visual effect takes place. This effect happens independent of whether the constraints added/removed are interesting (annotated) constraints.

B. Producing Animations

In the case of the CHR implementation of the tableaux proof technique, the main aim is to annotate every `branch/3` constraint with a graphical node object. As shown in Figure 1, each `branch(BNum,Level,E)` is annotated with a node having a green background. The name of the node is the result of concatenating the branch number and the level. The text inside the node is the expression `E`.

In addition the rule `oring` is annotated with two edge graphical objects between the node containing the ored expression and the two new nodes. In this case, the rule `oring` is annotated with a constraint `e(OldNode,NewNode)` where `OldNode` and `NewNode` contain the two identifiers. The constraint `e/2` is then annotated with an edge object.

An example of the visualization is given in Figure 2(a).

Another annotation example would affect the result of the visualization. In this case, the rule `branchF` could be annotated with the action `updateNode` introduced in [15] to change the color of the two nodes that cause the branch to fail to the color pink. In this case, the new visualizaed tree is shown in Figure 2(b).

In Figure 2(c), a different annotation was given. In this annotation the constraint `expression` produces a node with the remaining conjuncts as long as there are more than one.

Since the branch fails due to the existence of the two expressions a and $\neg(a)$, the rest of the tree is not shown.

Another example is given in Figure 3, in addition to the basic annotations, the rules `branchF2` and `branchF3` were similarly annotated to highlight the contradicting nodes in the sub-branch only.

As seen from the previous examples, throughout changing the annotations, different visualizations were produced. This could thus allow an instructor to adjust the visualization in the way they see suitable and most fitting to the taught material.

VI. Conclusions & Future Work

The tool presented enables students to see how expressions get evaluated with the tableaux techniques visually. The engine behind the platform uses annotation rules to abstract the details of the visualization and the proof production. The annotations could thus be pre-set for one time and used by all the students. Annotations could be also provided through an interface to eliminate the need of knowing any of the details. In the future, the platform should be extended with first-order logic constructs. Variables should be also handled and different types of expressions are to be explored.

References

[1] M. Ben-Ari, *Propositional Logic: Formulas, Models, Tableaux*. London: Springer London, 2012, pp. 7–47. [Online]. Available: https://doi.org/10.1007/978-1-4471-4129-7_2

[2] R. Hähnle and B. Beckert, "Proof confluent tableau calculi," in *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '99, Saratoga Springs, NY, USA, June 7-11, 1999, Proceedings*, ser. Lecture Notes in Computer Science, N. V. Murray, Ed., vol. 1617. Springer, 1999, pp. 34–35. [Online]. Available: https://doi.org/10.1007/3-540-48754-9_7

[3] B. Beckert and R. Goré, "Free-variable tableaux for propositional modal logics," *Studia Logica*, vol. 69, no. 1, pp. 59–96, 2001. [Online]. Available: https://doi.org/10.1023/A:1013886427723

[4] P. R. C. P. Maravic-Cisar Sanja I, Radosav Dragica, "Effectiveness of program visualization in learning java: a case study with jeliot 3," *International Journal of Computers Communications & Control*, vol. 6, no. 4, pp. 669–682, 2011.

[5] C. Hundhausen, S. Douglas, and J. Stasko, "A Meta-Study of Algorithm Visualization Effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[6] N. Sharaf, S. Abdennadher, and T. W. Frühwirth, "Chranimation: An animation tool for constraint handling rules," in *Logic-Based Program Synthesis and Transformation - 24th International Symposium, LOPSTR 2014, Canterbury, UK, September 9-11, 2014. Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Proietti and H. Seki, Eds., vol. 8981. Springer, 2014, pp. 92–110. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17822-6_6

[7] ——, "A rule-based approach for animating java algorithms," in *20th International Conference Information Visualisation, IV 2016, Lisbon, Portugal, July 19-22, 2016*, E. Banissi, M. W. M. Bannatyne, F. Bouali, R. Burkhard, J. Counsell, U. Cvek, M. J. Eppler, G. G. Grinstein, W. Huang, S. Kernbach, C. Lin, F. Lin, F. T. Marchese, C. M. Pun, M. Sarfraz, M. Trutschl, A. Ursyn, G. Venturini, T. G. Wyeld, and J. J. Zhang, Eds. IEEE Computer Society, 2016, pp. 141–145. [Online]. Available: https://doi.org/10.1109/IV.2016.55

[8] ——, "Using rules to animate prolog programs," in *Proceedings of the Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters @ RuleML+RR 2017 hosted by International Joint Conference on Rules and Reasoning 2017 (RuleML+RR 2017), London, UK, July 11-15, 2017.*, ser. CEUR Workshop Proceedings, N. Bassiliades, A. Bikakis, S. Costantini, E. Franconi, A. Giurca, R. Kontchakov, T. Patkos, F. Sadri, and W. V. Woensel, Eds., vol. 1875. CEUR-WS.org, 2017. [Online]. Available: http://ceur-ws.org/Vol-1875/paper24.pdf

[9] M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, *Handbook of tableau methods*. Springer Science & Business Media, 2013.

[10] T. W. Frühwirth, "Theory and practice of constraint handling rules," *Journal of Logic Programming. Special Issue on Constraint Logic Programming*, vol. 37, no. 1âĂŞ3, pp. 95 – 138, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743106698100055

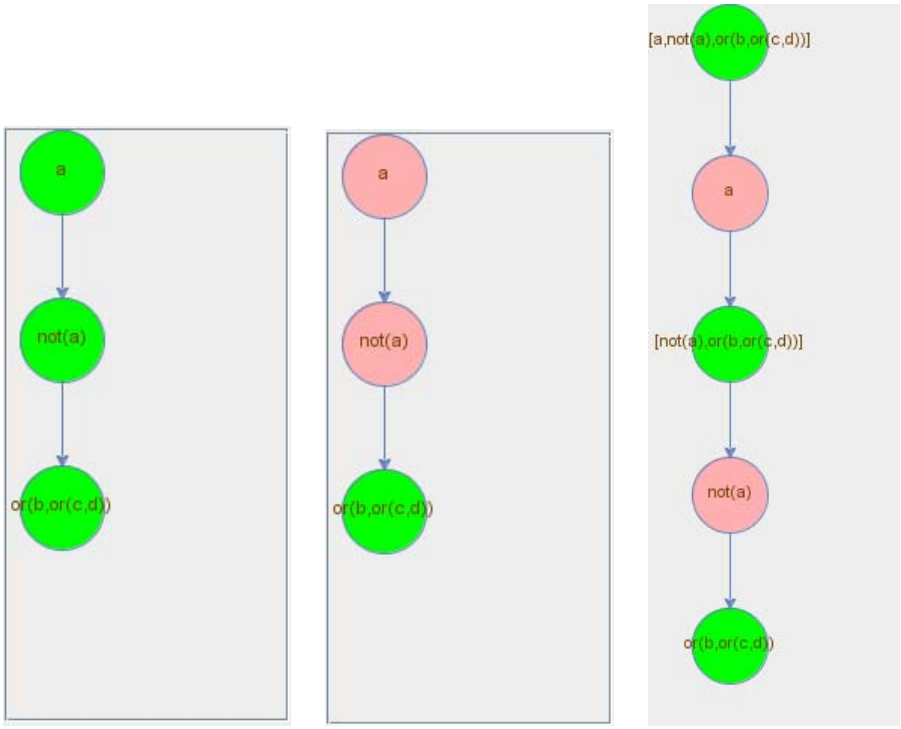
[11] T. Frühwirth, *Constraint Handling Rules*. Cambridge University Press, August 2009. [Online]. Available: http://www.constraint-handling-rules.org

[12] T. W. Frühwirth, "Constraint Handling Rules - What Else?" in *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Proceedings*, ser. Lecture Notes in Computer Science, N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, and D. Roman, Eds., vol. 9202. Springer, 2015, pp. 13–34. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21542-6_2

[13] T. W. Frühwirth and S. Abdennadher, *Essentials of constraint programming, chapter 9*, ser. Cognitive Technologies. Springer, 2003. [Online]. Available: http://www.springer.com/computer/swe/book/978-3-540-67623-2

[14] ——, "Principles of Constraint Systems and Constraint Solvers," *Archives of Control Sciences*, vol. 16, no. 2, pp. 131–159, 2006.

[15] N. Sharaf, S. Abdennadher, and T. W. Frühwirth, "Chr-graph: A platform for animating tree and graph algorithms," in *21st International Conference Information Visualisation, IV 2017, London, United Kingdom, July 11-14, 2017*. IEEE Computer Society, 2017, pp. 450–453. [Online]. Available: https://doi.org/10.1109/iV.2017.58

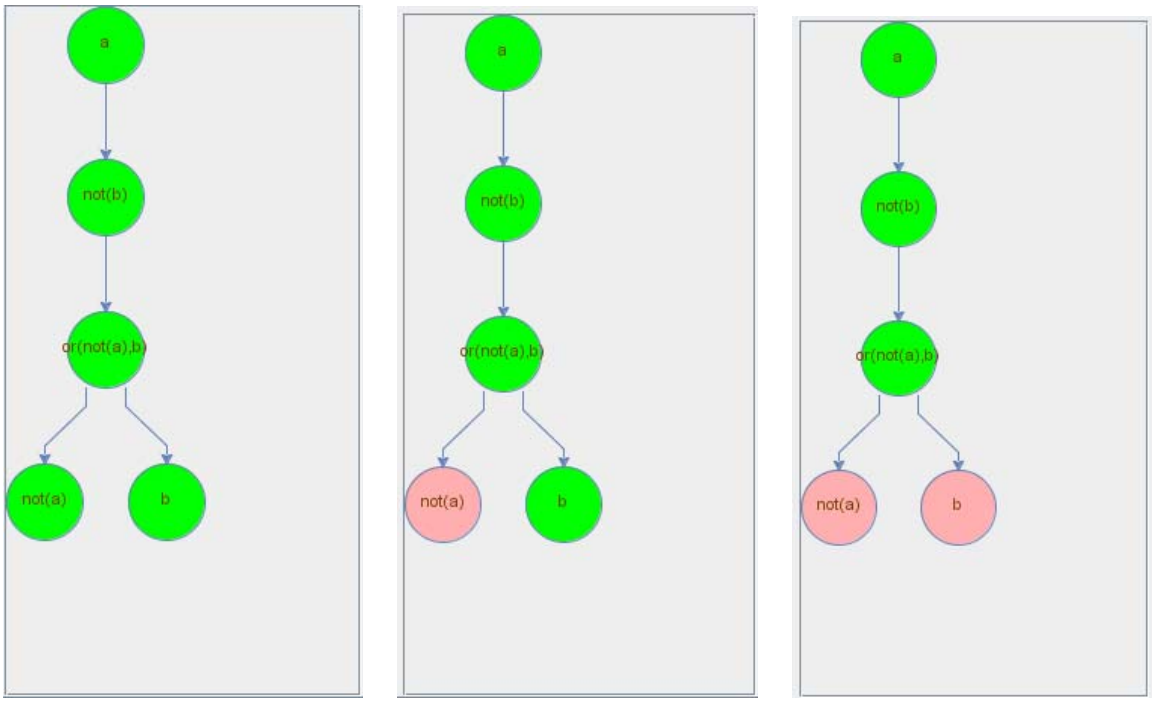


(a) Output

(b) Output with different annotation

(c) Output with another different annotation

Fig. 2. Example



(a) Example with Branching

(b) Failed sub-branch

(c) Another Failed sub-branch

Fig. 3. Another Example