Automated Reasoning*

Marek Kosta Christoph Weidenbach

Summer Term 2012

What is Computer Science about?

Theory

Graphics

Data Bases

Programming Languages

Algorithms

Hardware

Bioinformatics

Verification

What is Automated Deduction about?

Generic Problem Solving by a Computer Program.

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

Introductory Example: Solving 4×4 Sudoku

2	1		
		3	1
1		2	

Start

2	1	4	3
3	4	1	2
4	2	3	1
1	3	2	4

Solution

Formal Model

Represent board by a function $f(x, y)$ mapping cells to their value.

2	1		
		3	1
1		2	

Start

$N = f(1,1) \approx 2 \wedge f(1,2) \approx 1 \wedge$
$f(3,3) \approx 3 \wedge f(3,4) \approx 1 \wedge$
$f(4,1) \approx 1 \wedge f(4,3) \approx 2$

$\wedge$ is conjunction and $\top$ the empty conjunction.

A state is described by a triple $(N; D; r)$ where

- N contains the equations for the starting Sudoku
- D a conjunction of further equations computed by the algorithm
- $r \in \{\top, \bot\}$

Initial state is $(N; \top; \top)$.

A square $f(x, y)$ where $x, y \in \{1, 2, 3, 4\}$ is called *defined* by $N \wedge D$ if there is an equation $f(x, y) \approx z$, $z \in \{1, 2, 3, 4\}$ in N or D. For otherwise $f(x, y)$ it is called *undefined*.

Rule-Based Algorithm

Deduce $(N; D; \top) \quad \rightarrow \quad (N; D \wedge f(x,y) \approx 1; \top)$

provided $f(x,y)$ is undefined in $N \wedge D$, for any $x, y \in \{1,2,3,4\}$.

Conflict $(N; D; \top) \quad \rightarrow \quad (N; D; \bot)$

provided for $y \neq z$ (i) $f(x,y) = f(x,z)$ for $f(x,y)$, $f(x,z)$ defined in $N \wedge D$ for some x, y, z or (ii) $f(y,x) = f(z,x)$ for $f(y,x)$, $f(z,x)$ defined in $N \wedge D$ for some x, y, z or (iii) $f(x,y) = f(x',y')$ for $f(x,y)$, $f(x',y')$ defined in $N \wedge D$ and $[x, x' \in \{1,2\}$ or $x, x' \in \{3,4\}]$ and $[y, y' \in \{1,2\}$ or $y, y' \in \{3,4\}]$ and $x \neq x'$ or $y \neq y'$.

Backtrack $(N; D' \wedge f(x,y) \approx z \wedge D''; \bot) \quad \rightarrow \quad (N; D' \wedge f(x,y) \approx z+1; \top)$

provided $z < 4$ and $D'' = \top$ or D'' contains only equations of the form $f(x',y') \approx 4$.

Fail $(N; D; \bot) \quad \rightarrow \quad (N; \top; \bot)$

provided $D \neq \top$ and D contains only equations of the form $f(x,y) \approx 4$.

Properties: Rules are applied don't care non-deterministically.

An algorithm (set of rules) is *sound* if whenever it declares having found a solution it actually has computed a solution.

It is *complete* if it finds a solution if one exists.

It is *terminating* if it never runs forever.

Proposition 0.1 (Soundness) *The rules Deduce, Conflict, Backtrack and Fail are sound. Starting from an initial state* $(N; \top; \top)$*:*
(i) for any final state $(N; D; \top)$*, the equations in* $N \wedge D$ *are a solution, and,*
(ii) for any final state $(N; \top; \bot)$ *there is no solution to the initial problem.*

Proof. (i) So assume a final state $(N; D; \top)$ such that no rule is applicable. In particular, this means that for all $x, y \in \{1,2,3,4\}$ the square $f(x,y)$ is defined in $N \wedge D$ as for otherwise Deduce would be applicable, contradicting that $(N; D; \top)$ is a final state. So all squares are defined by $N \wedge D$. What remains to be shown is that those assignments actually constitute a solution to the Sudoku. However, if some assignment in $N \wedge D$ results in a repetition of a number in some column, row or 2×2 box of the Sudoku, then rule Conflict is applicable, contradicting that $(N; D; \top)$ is a final state. In sum, $(N; D; \top)$ is a solution to the Sudoku and hence the rules Deduce, Conflict, Backtrack and Fail are sound.

(ii) So assume that the initial problem $(N; \top; \top)$ has a solution. We prove that in this case we cannot reach a state $(N; \top; \bot)$. Let $(N; D; \top)$ be an arbitrary state still having a solution. This includes the initial state if $D = \top$. We prove that we can correctly decide the next square. Since $(N; D; \top)$ still has a solution the only applicable rule is Deduce and we generate $(N; D \wedge f(x, y) \approx 1; \top)$ for some $x, y \in \{1, 2, 3, 4\}$. If $(N; D \wedge f(x, y) \approx 1; \top)$ still has a solution we are done. So assume $(N; D \wedge f(x, y) \approx 1; \top)$ does not have a solution anymore. But then eventually we will apply Conflict and Backtrack to a state $(N; D \wedge f(x, y) \approx 1 \wedge D'; \bot)$ where D' only contains equations of the form $f(x', y') \approx 4$ resulting in $(N; D \wedge f(x, y) \approx 2; \top)$. Now repeating the argument we will eventually reach a state $(N; D \wedge f(x, y) \approx k; \top)$ that has a solution.

Proposition 0.2 (Completeness) *The rules Deduce, Conflict, Backtrack and Fail are complete. For any solution $N \wedge D$ of the Sudoku there is a sequence of rule applications such that $(N; D; \top)$ is a final state.*

Proposition 0.3 (Termination) *The rules Deduce, Conflict, Backtrack and Fail terminate on any input state $(N; \top; \top)$.*

Confluence

Another important property for don't care non-deterministic rule based definitions of algorithms is *confluence*.

It means that whenever several sequences of rules are applicable to a given states, the respective results can be rejoined by further rule applications to a common problem state.

Proposition 0.4 (Deduce and Conflict are Locally Confluent) *Given a state $(N; D; \top)$ out of which two different states $(N; D_1; \top)$ and $(N; D_2; \bot)$ can be generated by Deduce and Conflict, respectively, then the two states can be rejoined to a state $(N; D'; *)$ via further rule applications.*

Result

It works.

But: It looks like a lot of effort for a problem that one can solve with a little bit of thinking.

Reason: Pupils learn not only axioms or rules, but also recipes to work efficiently with these.

This difference is also important for automated reasoning: