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.*

Proof . A particular strategy for the rule applications is needed to indeed generate $(N; D; \top)$ out of $(N; \top; \top)$ for some solution $N \wedge D$. Without loss of generality we assume the assignments in D to be sorted such that assignments to a number $k \in \{1, 2, 3, 4\}$ precede any assignment to some number $l > k$. So if, for example, N does not assign all four values 1, then the first assignment in D is of the form $f(x, y) \approx 1$ for some x, y. Now we apply the following strategy, subsequently adding all assignments from D to $(N; \top; \top)$. Let us assume we have already achieved state $(N; D'; \top)$ and the next assignment from D to be established is $f(x, y) \approx k$, meaning $f(x, y)$ is not defined in $N \wedge D'$. Then until $l = k$ we do the following, starting from $l = 1$. We apply Deduce adding the assignment $f(x, y) \approx l$. If Conflict is applicable to this assignment, we apply Backtrack, generating the new assignment $f(x, y) \approx l + 1$ and continue.

We need to show that this strategy in fact eventually adds $f(x, y) \approx k$ to D'. As long as $l < k$ any added assignment $f(x, y) \approx l$ results in rule Conflict applicable, because D is ordered and all four values for all $l < k$ are already established. The eventual assignment $f(x, y) \approx k$ does not generate a conflict because D is a solution. For the same reason, the rule Fail is never applicable. Therefore, the strategy generates $(N; D; \top)$ out of $(N; \top; \top)$.

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

Proof . Once the rule Fail is applicable, no other rule is applicable on the result anymore. So we do not need to consider rule Fail for termination. The idea of the proof is that we assign a *measure* over the natural numbers to every state such that each rule strictly decreases the measure and that the measure cannot get below 0. For any given state $S = (N; D; *)$ with $* \in \{\top, \bot\}$ with $D = f(x_1, y_1) \approx k_1 \wedge \ldots \wedge f(x_n, y_n) \approx k_n$ we

assign the measure $\phi(S)$ by $\phi(S) = 2^{49} - p - \sum_{i=1}^{n} k_i \cdot 2^{49-3i}$ where $p = 0$ if $* = \top$ and $p = 1$ otherwise.

The measure $\phi(S)$ is well-defined and cannot become negative as $n \leq 16$, $p \leq 1$, and $1 \leq k_i \leq 4$ for any D. In particular, the former holds because the rule Deduce only adds values for undefined squares and the overall number of squares is bound to 16. What remains to be shown is that each rule application decreases ϕ. We do this by a case analysis over the rules.

Deduce: $\phi((N; D; \top)) = 2^{49} - \sum_{i=1}^{n} k_i \cdot 2^{49-3i} > 2^{49} - \sum_{i=1}^{n} k_i \cdot 2^{49-3i} - 1 \cdot 2^{49-3(n+1)} = \phi((N; D \wedge f(x,y) \approx 1; \top))$

Conflict: $\phi((N; D; \top)) = 2^{49} - \sum_{i=1}^{n} k_i \cdot 2^{49-3i} > 2^{49} - 1 - \sum_{i=1}^{n} k_i \cdot 2^{49-3i} = \phi((N; D; \bot))$

Backtrack: $\phi((N; D' \wedge f(x_l, y_l) \approx k_l \wedge D''; \bot)) = 2^{49} - 1 - (\sum_{i=1}^{l-1} k_i \cdot 2^{49-3i}) - k_l \cdot 2^{49-3l} - \sum_{i=l+1}^{n} k_i \cdot 2^{49-3i} > 2^{49} - (\sum_{i=1}^{l-1} k_i \cdot 2^{49-3i}) - (k_l + 1) \cdot 2^{49-3l} = \phi(N; D' \wedge f(x_l, y_l) \approx k_l + 1; \top)$ in particular because $2^{49-3l} > \sum_{i=l+1}^{n} k_i \cdot 2^{49-3i} + 1$.

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 in one step, respectively, then the two states can be rejoined to a state $(N; D'; *)$ via further rule applications.*

Proof . Consider an application of Deduce and Conflict to a state $(N; D; \top)$ resulting in $(N; D \wedge f(x,y) \approx 1; \top)$ and $(N; D; \bot)$, respectively. We will now show that in fact we can rejoin the two states. Notice that since Conflict is applicable to $(N; D; \top)$ it is also applicable to $(N; D \wedge f(x,y) \approx 1; \top)$. So the first sequence of rejoin steps is

$$\begin{array}{rcl} (N; D \wedge f(x,y) \approx 1; \top) & \rightarrow & (N; D \wedge f(x,y) \approx 1; \bot) \\ & \rightarrow & (N; D \wedge f(x,y) \approx 2; \top) \\ & \rightarrow^* & (N; D \wedge f(x,y) \approx 4; \bot) \end{array}$$

where we subsequently applied Conflict and Backtrack to reach the state $(N; D \wedge f(x,y) \approx 4; \bot)$ and $\rightarrow^*$ abbreviates those finite number of rule applications. Finally applying Backtrack (or Fail) to $(N; D; \bot)$ and $(N; D \wedge f(x,y) \approx 4; \bot)$ results in the same state.

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: Our approach is very general, it can actually be used to "pontentially solve" *any* problem in computer science.

This difference is also important for automated reasoning:

- For problems that are well-known and frequently used, we can develop optimal specialized methods.
  ⇒ Algorithms & Data-structures

- For new/unknown/changing problems, we have to develop generic methods that do "something useful".
  ⇒ this lecture: Logic + Calculus + Implementation
- Combining the two approaches
  ⇒ Automated Reasoing II (next semester): Logic modulo Theory + Calculus + Implementation

Topics of the Course

Preliminaries

math repetition
computer science repetition
orderings
induction (repetition)
rewrite systems

Propositional logic

logic: syntax, semantics
calculi: superposition, CDCL
implementation: 2-watched literal, clause learning

First-order predicate logic

logic: syntax, semantics, model theory
calculus: superposition
implementation: sharing, indexing

First-order predicate logic with equality

equational logic: unit equations
calculus: term rewriting systems, Knuth-Bendix completion
implementation: dependency pairs
first-order logic with equality
calculus: superposition
implementation: rewriting

Literature

Is a big problem, actually you are the "guinea-pigs" for a new textbook.

Franz Baader and Tobias Nipkow: *Term rewriting and all that*, Cambridge Univ. Press, 1998. (Textbook on equational reasoning)

Armin Biere and Marijn Heule and Hans van Maaren and Toby Walsh (editors): *Handbook of Satisfiability*, IOS Press, 2009. (Be careful: Handbook, hard to read)

Alan Robinson and Andrei Voronkov (editors): *Handbook of Automated Reasoning*, Vol I & II, Elsevier, 2001. (Be careful: Handbook, very hard to read)

1 Preliminaries

- math repetition
- computer science repetition
- orderings
- induction (repetition)
- rewrite systems

1.1 Mathematical Prerequisites

$\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers

$\mathbb{N}^+$ is the set of positive natural numbers without 0

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ denote the integers, rational numbers and the real numbers, respectively.

Multisets

Given a set M, a *multi-set* S over M is a mapping $S\colon M \to \mathbb{N}$, where S specifies the number of occurrences of elements m of the base set M within the multiset S.

We use the standard set notations $\in$, $\subset$, $\subseteq$, $\cup$, $\cap$ with the analogous meaning for multisets, e.g., $(S_1 \cup S_2)(m) = S_1(m) + S_2(m)$.

We also write multi-sets in a set like notation, e.g., the multi-set $S = \{1, 2, 2, 4\}$ denotes a multi-set over the set $\{1, 2, 3, 4\}$ where $S(1) = 1$, $S(2) = 2$, $S(3) = 0$, and $S(4) = 1$.

A multi-set S over a set M is *finite* if $\{m \in M \mid S(m) > 0\}$ is finite. In this lecture we only consider finite multi-sets.

Relations

An n-ary *relation* R over some set M is a subset of M^n: $R \subseteq M^n$.

For two n-ary relations R, Q over some set M, their union ($\cup$) or intersection ($\cap$) is again an n-ary relation, where
$R \cup Q := \{(m_1, \ldots, m_n) \in M \mid (m_1, \ldots, m_n) \in R \text{ or } (m_1, \ldots, m_n) \in Q\}$
$R \cap Q := \{(m_1, \ldots, m_n) \in M \mid (m_1, \ldots, m_n) \in R \text{ and } (m_1, \ldots, m_n) \in Q\}$.

A relation Q is a *subrelation* of a relation R if $Q \subseteq R$.

The *characteristic function* of a relation R or sometimes called *predicate* indicates membership. In addition of writing $(m_1, \ldots, m_n) \in R$ we also write $R(m_1, \ldots, m_n)$. So the predicate $R(m_1, \ldots, m_n)$ holds or is true if in fact $(m_1, \ldots, m_n)$ belongs to the relation R.

Words

Given a nonempty alphabet Σ the set Σ^* of *finite words* over Σ is defined by

(i) the empty word $\epsilon \in \Sigma^*$

(ii) for each letter $a \in \Sigma$ also $a \in \Sigma^*$

(iii) if $u, v \in \Sigma^*$ so $uv \in \Sigma^*$ where uv denotes the concatenation of u and v.

The length $|u|$ of a word $u \in \Sigma^*$ is defined by

(i) $|\epsilon| := 0$,

(ii) $|a| := 1$ for any $a \in \Sigma$ and

(iii) $|uv| := |u| + |v|$ for any $u, v \in \Sigma^*$.