Constraint Reasoning and Programming with CHR

Thom Frühwirth
Ludwig-Maximilians-University Munich
Department of Computer Science
Oettingenstrasse 67, D-80538 Munich Germany
fruehwir@informatik.uni-muenchen.de
http://www.pst.informatik.uni-muenchen.de/~fruehwir/

Tutorial at ISLIP'99

Anmerkung

Constraint Reasoning and Constraint Programming

A generic framework for
Modelling
- with partial information
- with infinite information
Reasoning
- with new information
Solving
- combinatorial problems

Anmerkung

Constraint Reasoning

The Idea

- Example numeric lock:

  0 1 2 3 4 5 6 7 8 9

  Greater or equal 5.

  Prime number.
- Declarative problem representation by variables and constraints:

  $x \in \{0, 1, \ldots, 9\} \wedge x \geq 5 \wedge prime(x)$
- Constraint propagation and simplification reduce search space:

  $x \in \{0, 1, \ldots, 9\} \wedge x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\}$

Anmerkung

- Getrennt von Problemlösungsstrategie:
  Konjunktion von atomaren Formeln in Prädikatenlogik erster Stufe mit Gleichheit.
- Logische Folgerungen.

Sieht trivial aus, Probleme sind aber oft **NP-vollständig**, Algorithmen exponentiell.

Constraints = Einschränkungen, (Wert-, Rand-, Neben-)Bedingungen.

Constraints beschreiben die Eigenschaften und Beziehungen von teilweise unbekannten Objekten.

(Atomare) Constraints: Spezielle Prädikate der Prädikatenlogik erster Stufe.

Constrainttheorie: Nichtleere, konsistente Theorie, die die Constraints beschreibt.

Constraint(problem), Anfrage: Zu lösende, allquantorfreie Konjunktion von atomaren Constraints.

Lösung, Antwort: Wertebelegungen für Variablen, die Anfrage erfüllen.
Allgemeiner: Möglichst starke Vereinfachung (z.B. Normalform) einer Anfrage.

Constraintlöser: Programm, das Constraints vereinfacht und löst.

Constraint-Programm: schickt nach und nach Constraints an Constraintlöser, erwartet (Teil-)Lösungen.

Constraint Programming

Some Applications

- Lufthansa: Short term personell planning.
- Hongkong: Container Harbor resource planning.
- Renault: Short term production planning.
- NASA: Hubble space telescope experiment scheduling.
- Airbus: Cabin layout.
- Siemens: Circuit verification.
- Nokia: Software configuration for mobile phones.
- Caisse d'epargne: Portfolio management.

In Decision Support Systems for
Planning, Konfiguration, for Design, Analysis.

[Frühwirth, Abdennadher, *Constraint-Programmierung*, Springer Verlag, 1997]

Anmerkung

Umsatz 1996 ca. 100 Millionen Dollar, wie Data Mining, 1% von Microsoft. Kommerziellen Anwendungen 1996 ca. 300.

- Ilog (F,USA): Ilog Optimization Suite (incl. C-Plex, Numerica).
- Cosytec (F): CHIP V5.
- Siemens, IF Computer (A,J,D): IF/Prolog V5.1.
- PrologIA (F): Prolog IV.

- Lufthansa: DAYSY, Cosytec CHIP.
- Hongkong International Terminals: Transport und Lagerung von Containern. ICL DecisionPower. Ablaufplanung = Scheduling
- Renault: Bull CHIP-Derivat.
- Airbus: Uni Hamburg.
- Siemens: IFProlog.
- Caisse d'epargne: PrologIA
- Daussault.
- SNCF 1700 Züge Gare Du Nord Paris Ilog.
- Venice lagoon pollution warch

Personal- und Ablaufplanung, Transport- und Plazierungsoptimierung.

Gelbe Seiten Layout: Anzeigen und Einträge.
A+E nahe, Konkurrenten weg, Seite voll.

Zahlenschloß, Kreuzworträtsel, Gleichungslösen.

Musikprobe - Lärmschutzvorschriften.

Application Domains

Modelling, Executable Specification.

Solving combinatorial problems:

- Scheduling, Planning, Timetabling
- Configuration, Layout, Placement, Design
- Analysis: Simulation, Verification, Diagnosis

of software, electronic, electrical, mechanical hardware components and industrial processes.

Artificial Intelligence

- Machine Vision
- Natural Language Understanding
- Temporal and Spatial Reasoning
- Theorem Proving
- Qualtitative Reasoning

Anmerkung

Finance and Banking
Moleacular Biology, Genome Mapping

History

60ties,70ties Constraint networks in artificial intelligence.
70ties Logic programming (Prolog).
80ties Constraint logic programming.
80ties Concurrent logic programming.
90ties Concurrent constraint programming.
90ties Commercial applications.

Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

[Eugene C. Freuder, *Constraints Journal*, 1997]

Anmerkung

Constraint Programming

Robust, flexible, maintainable software faster.

- Declarative modelling by constraints:
  Description of the properties and relationships between partially unknown objects.
  Correct handling of precise and imprecise, partial and full information.
  $x \in \{0, 1, \ldots, 9\} \wedge x \geq 5 \wedge prime(x)$
- Automatic constraint reasoning:
  Propagation of the effects of new information.
  Simplification makes implicit information explicit.
  $x \in \{0, 1, \ldots, 9\} \wedge x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\}$
- Solving combinatorial problems efficiently:
  Easy Combination of constraint solving with search and optimization.

Anmerkung

Constraints = Einschränkungen, (Wert-, Rand-, Neben-)Bedingungen.

Anpassung und Kombination existierender Verfahren:

- Mathematik: Operations Research
- Informatik: KI, Graphentheorie
- Betriebswirtschaft: Ablaufplanung
- Linguistik: Feature Terme
- Algebra, Endliche Automaten, Automatisches Beweisen...

Terminology

Our language is first order logic.

Constraint
Conjunction of atomic constraints (predicates)
E.g. $4X + 3Y = 10 \wedge 2X - Y = 0$

Constraint Problem (Query)
A given, initial constraint

Constraint Solution (Answer)
A valuation for the variables in a given constraint problem that satisfies all constraints of the problem
E.g. $X = 1 \wedge Y = 2$
In general, a normal/solved form of a constraint problem
E.g. $4X + 3Y + Z = 10 \wedge 2X - Y = 0$ simplifies into $Y + Z = 10 \wedge 2X - Y = 0$

Anmerkung

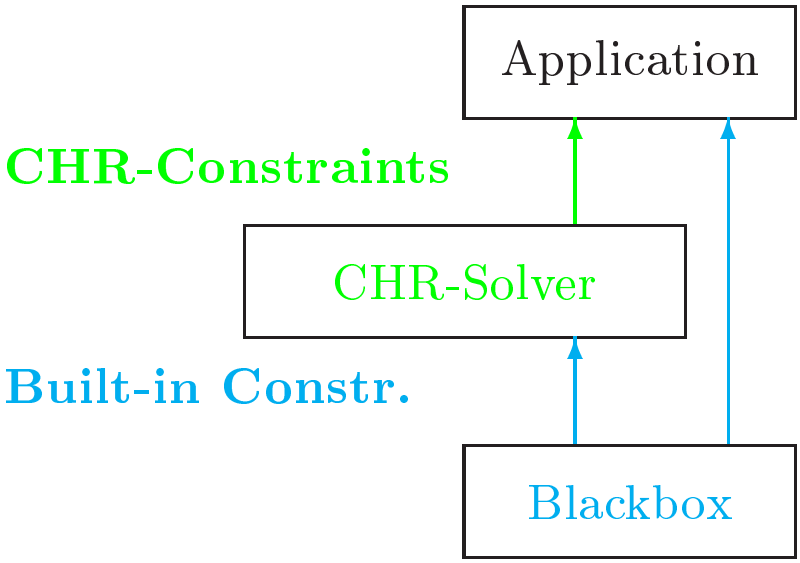
Transform a constraint problem into a logically equivalent, but somehow simpler form.

Present Principles, a FRAMEWORK.

Constraint Handling Rules (CHR)

Declarative programming language for the specification and implementation of constraint solvers and programs.

CHR constraint solvers are open and flexible, can be maintained, combined, debugged, analysed.



Host language (Prolog, Lisp, Java,...)

[Frühwirth, *Journal of Logic Programming*, 1998]

Anmerkung

Extremster Ansatz: Eigene Programmiersprache.

Constraint-Programm: schickt nach und nach Constraints an Constraintlöser, erwartet (Teil-)Lösungen.

Constraintlöser: Programm, das Constraints propagiert, vereinfacht und löst.

Built-in Constraints (z.B. auch Tests und Zuweisungen)

Vorteile Constraint Handling Rules

- Höhere Programmiersprache
- Deklarative Semantik
- Theoretische Resultate
- Konkrete Anwendungen
- Programmanalyse
- Neue Constraints möglich

Example Order Relation $\leq$

$x \leq x \Leftrightarrow true$	Reflexivity
$x \leq y \wedge y \leq x \Leftrightarrow x = y$	Antisymmetry
$x \leq y \wedge y \leq z \Rightarrow x \leq z$	Transitivity
$a \leq b \wedge b \leq c \wedge c \leq a$	*Query*
	Transitivity
$a \leq c$	
	Antisymmetry
$a = c$	Blackbox Solver
	Antisymmetry
$a = b$	
$a = b \wedge a = c$	*Answer*

Anmerkung

Constraint aus Zahlenschloß-Beispiel.

`x, y, z` sowie `a, b, c` sind ungebundene Variablen.

Linie zwischen Program und Query ziehen.

Constraint-Problem heißt hier Query,
Lösung bzw. möglichst vereinfachte Form heißt hier Answer.

GLEICH BEISPIEL MACHEN!

Constraints sind (allquantorfreie) Konjunktionen von atomaren Constraints.

CHR-Regeln beschreiben wie Constraints sich zu neuen Constraints vereinfachen und wie Constraints andere Constraints propagieren.

Fortgesetzte Vereinfachung und Propagierung von Constraints führt schließlich zur Lösung der Constraints.

Regeln aus Inferenzregeln, Formeln, Ersetzungsregeln gewinnbar.

Hier Abstrakte Syntax verwendet.

Simplifikation entfernt Constraints, Propagierung nicht.

Kein Beweiser, sondern effizient Konjunktionen lösen.

Beispiel mit Wächter: Reflexivity mit Wächter.

Declarative Semantics

Head	K	CHR-Constraints
Guard	W	Built-in Constraints
Body	R	Built-in and CHR-Constraints

Simplification rule

$K \Leftrightarrow W \mid R \qquad \forall\ (W \rightarrow (K \leftrightarrow \exists \bar{x}\ R))$

Propagation rule

$K \Rightarrow W \mid R \qquad \forall\ (W \rightarrow (K \rightarrow \exists \bar{x}\ R))$

Declarative semantics of a CHR program P:

$\mathcal{P}$: Logical reading of the CHR rules of P.

$\cup\ \mathcal{T}$: Theory for built-in constraints.

Anmerkung

Meta-Variablen stehen immer für Konjunktionen von Constraints.

Variablen $\bar{x}$ kommen *nur* in R vor.

Constraints als spezielle Prädikate der Prädikatenlogik erster Stufe.
Built-in: *true*, *false* = syntaktische Gleichheit.

CHR-Constraints durch Regeln definiert und implementiert.

Der Wächter als Vorbedingung einer Regel impliziert dabei einen logischen Zusammenhang zwischen Kopf und Rumpf der Regel: Im Fall einer Simplifikationsregel ist der Zusammenhang eine logischen Äquivalenz zwischen Kopf und Rumpf, im Fall einer Propagierungsregel eine Implikation.

Mit Wächter:

$\forall$`X,Y (X`$=$`Y`$\rightarrow$`(X`$\leq$`Y`$\leftrightarrow$`true)) % Reflexivity`

Operational Semantics

State Transition System

State: Conjunction of constraints.

Simplify $A \wedge C \longmapsto R\sigma \wedge C$
if $(K \Leftrightarrow W \mid R) \in P$
and $(A = K\sigma)$ and $(C_V \rightarrow W\sigma)$

Propagate $A \wedge C \longmapsto R\sigma \wedge A \wedge C$
if $(K \Rightarrow W \mid R) \in P$
and $(A = K\sigma)$ and $(C_V \rightarrow W\sigma)$

Split $(A \vee B) \wedge C \longmapsto (A \wedge C) \vee (A \wedge C)$

Anmerkung

K, W, R, C: Konjunktionen von Constraints.

Substitution, Matching, Instanz

Jede passende CHR-Regel-Kopie auswählbar.
committed-choice. verpflichtende (Regel-)wahl
Matching anstatt Unifikation (sonst incomplete).
Wächter zur Regelauswahl (da kein backtracking).

$\bar{x}$ frische Variablen aus Regelkopie.

Vereinfachen von built-in Constraints implizit.

Konjunktion assoziativ und kommutativ, nicht idempotent.

$\forall$ Allabschluß.

W im Folgezustand nicht dabei.

Terminierung der Propagierungsregeln nicht dargestellt.

Correspondence between Semantics

Lemma (Equivalence of States) If the state B appears in a computation of A, then

$$\mathcal{P}, \mathcal{T} \models \forall\ (A \leftrightarrow \exists \bar{x} B).$$

Theorem (Soundness) If C is an answer of A, then

$$\mathcal{P}, \mathcal{T} \models \forall\ (A \leftrightarrow \exists \bar{x} C).$$

Theorem (Completeness) A has at least one finite computation. If $\mathcal{P}, \mathcal{T} \models \forall\ (A \leftrightarrow \exists \bar{x} C)$, then A has an answer C' such that

$$\mathcal{P}, \mathcal{T} \models \forall\ (\exists \bar{x} C \leftrightarrow \exists \bar{y} C').$$

[Abdennadher, Frühwirth, Meuss, *Constraint Journal*, Kluwer, 1999]

Anmerkung

Ableitung: Folge von Reduktionen.

$\exists \bar{x}, y$: Alle Variablen, die nicht in A vorkommen.

Sätze besser als für herkömmliche CLP-Sprachen.

Vollständigkeits- und Korrektheitssätze des CHR-Kalküls basieren vor allem darauf, daß Reduktionen für CHR die logische Äquivalenz von den Zuständen erhalten:

Induktionsbeweis über Länge der Ableitungen plus logische Leseweise der Reduktionsregeln.

Aus diesem Lemma folgt unmittelbar, daß alle Zustände in einer Ableitung einer Query logisch äquivalent sind.

Aus dem Lemma folgt auch die Korrektheit.

Der Vollständigkeitssatz gilt nicht, wenn A keine endlichen Ableitungen hat:

$$\mathtt{p} \;\Leftrightarrow\; \mathtt{p}$$

Sei A die Query $\mathtt{p}$. Es gilt $\mathcal{P}, \mathcal{T} \models \mathtt{p} \leftrightarrow \mathtt{p}$.
Aber A hat nur eine unendliche Ableitung.

Vollständigkeit schwach für Answer *false*:

$$\mathtt{p} \;\Leftrightarrow\; \mathtt{q}$$
$$\mathtt{p} \;\Leftrightarrow\; \mathit{false}$$

Es gilt $\mathcal{P}, CT \models \neg\mathtt{q}$. Aber $\mathtt{q}$ hat nur Answer $\mathtt{q}$.

Program Analysis

Confluence

The answer of a query is always the same, no matter which of the applicable rules are applied.

$$x \leq y \wedge y \leq x \Leftrightarrow x = y$$
$$x \leq y \wedge y \leq z \Rightarrow x \leq z$$

$$x \leq y \wedge y \leq x$$

$$x \leq y \wedge y \leq x \wedge x \leq x \qquad x = y$$

$$x = y \wedge x \leq x \qquad x = y$$

Theorem (Confluence) A terminating CHR program is confluent iff its critical pairs are joinable.

[Abdennadher, Frühwirth, Meuss, *Constraint Journal*, Kluwer, 1999]

Anmerkung

A, B, C, D sind Zustände, d.h. Constraints.

Probleme bei der Adaptation von Ergebnissen aus Termersetzungssystemen (TRS):

- Propagationsregeln
- Blackbox-Constraintlöser

Monotonie entspricht *Verträglichkeit* bei TRS: Wenn $A \longmapsto B$, dann auch $A \wedge D \longmapsto B \wedge D$.

Überlappen zweier Regeln ergibt Ausgangszustand.

Ausgangszustand hat die Köpfe der beiden Regeln, wobei ein oder mehrere Kopfconstraints miteinander gleichgesetzt wurden, und die Wächter der beiden Regeln.

Entscheidbares, hinreichendes und notwendiges Kriterium.

Nutzen des Confluencetests:

Theorie:
Impliziert Konsistenz und verbessert Übereinstimmung der Semantics.

Praxis:
Answer auch unabhängig von Reihenfolge der Constraints.

Confluencetest enthüllt Fehler.
Implementierte CHR-Constraintlöser sind meist confluent.

Vervollständigung kann nicht-confluente Programme confluent machen.

Constraint Handling Rules

www.pst.informatik.uni-muenchen.de/∼fruehwir/

7 Implementations
- Eclipse 4.0 Prolog
- Sicstus 3.7 Prolog
- Java

30 Applications
- Theorem Proving, Uni York, Uni Koblenz
- Munich rent advisor, LMU Munich
- Placement of senders for wireless communication, LMU Munich
- Executable Z-Specifications, Uni Melbourne
- Morphological analysis of natural language, CIS Munich
- Integration and Brokering of Internet Information, MIT Boston
- Kanji Font Description, Uni Zurich
- Automatic Generation of Constraint Solvers, CWI Amsterdam
- University Lecture Timetabling and Roomplanning, LMU Munich

Anmerkung

CHR Constraint Solvers

50 Constraint Solvers
- Boolean Algebra, Propositional Logic, Uni York
- Terms, Rational Trees, Feature Trees, Types, Lists
- Finite Domains
- Linear Polynomial (In-)Equations
- Nonlinear Polynomial (In-)Equations, CRIN-CNRS and INRIA-Lorraine
- Interval Arithemtic
- Generic Arc and Path Consistency
- Temporal Reasoning, CISE Milano, Uni Linkoeping
- Spatial Reasoning, Uni Jaume I Casellun
- Terminological Reasoning, DFKI Kaiserslautern, Uni Bucharest, GMD FIRST Berlin
- Diagrammatical Reasoning, Uni Melbourne

Anmerkung

Boolean Constraints

$$
\begin{aligned}
X \sqcap Y = Z \wedge X = 0 &\Rightarrow Z = 0 \\
X \sqcap Y = Z \wedge Y = 0 &\Rightarrow Z = 0 \\
X \sqcap Y = Z \wedge X = 1 &\Rightarrow Y = Z \\
X \sqcap Y = Z \wedge Y = 1 &\Rightarrow X = Z \\
X \sqcap Y = Z \wedge Z = 1 &\Rightarrow X = 1 \wedge Y = 1 \\
X \sqcap Y = Z \wedge X = Y &\Rightarrow Y = Z
\end{aligned}
$$

$$
\begin{aligned}
&add(I1, I2, I3, O1, O2) \Leftrightarrow \\
&\quad xor(I1, I2, X1) \wedge and(I1, I2, A1) \wedge \\
&\quad xor(X1, I3, O2) \wedge and(I3, X1, A2) \wedge \\
&\quad or(A1, A2, O1)
\end{aligned}
$$

```
:- add(I1,I2,I3,[O1,O2]),I3=0,O1=1
I3=0,O1=1,I1=1,I2=1,O2=0
```

Consider the predicate `add/4` taken from the well-known full-adder circuit. It adds three single digit binary numbers to produce a single number consisting of two digits:

The computation proceeds as follows: Because `I3=0`, the output `A2` of the and-gate with input `I3` must be 0. As `O1=1` and `A2=0`, the other input `A1` of the or-gate must be 1. Because `A1` is also the output of an and-gate, its inputs `I1` and `I2` must be both 1. Hence the output `X1` of the first xor-gate must be 0, and therefore also the output `O2` of the second xor-gate must be 0. The query `add(1,1,I3,[O1,O2])` reduces to `I3=O2,O1=1`. This example illustrates the power of this simple but incomplete solver.

Theorem Proving

Resolution

$$cl(\{+X\} \cup L1) \wedge cl(\{-X\} \cup L2) \Rightarrow cl(L1 \cup L2)$$

Unit Resolution

$$\begin{aligned}
empty_cl \ @ \ cl(\{\}) &\Leftrightarrow false \\
tautology \ @ \ cl(\{+X, -X\} \cup L1) &\Leftrightarrow true \\
\\
unit_instantiation \ @ \ cl(\{+X\}) &\Leftrightarrow X = 1 \\
unit_instantiation \ @ \ cl(\{-X\}) &\Leftrightarrow X = 0 \\
unit_propagation \ @ \ cl(\{+0\} \cup L1) &\Leftrightarrow cl(L1) \\
unit_propagation \ @ \ cl(\{-1\} \cup L1) &\Leftrightarrow cl(L1) \\
unit_subsumption \ @ \ cl(\{+1\} \cup L1) &\Leftrightarrow true \\
unit_subsumption \ @ \ cl(\{-0\} \cup L1) &\Leftrightarrow true
\end{aligned}$$

In [Dum95] experiments were performed in applying resolution and backtracking to solving Boolean constraint satisfaction problems. The DP procedure restricts resolution to unit clauses. A labeling phase is added that tries truth values using backtracking for the variables one by one, thus retaining completeness.

Here is an incremental version of the DP procedure[1], other versions of resolution can also be found in [Dum95]. Boolean CSPs are modeled as conjunctions of clauses, where a clause is a disjunction of literals (positive or negative atomic propositions). A clause is represented as a list of signed Boolean variables. For example, $\neg a \vee b \vee c$ is represented as `cl({-A,+B,+C})`. The lists are closed, variables in the lists are ordered. `member/2` is the usual Prolog predicate about lists.

[1] "Pure literal deletion" is not implemented, because it is based on a global condition which is not sound anymore when constraints can be added incrementally as is the case in CHR.

n-Queens Predicates

```
solve(N,Qs) ⇔
   make_domains(N,Qs) ∧ queens(Qs) ∧ labeling(Qs).

queens(L) ⇔
   L = []
 ∨ L = [X|Xs] ∧ safe(X,Xs,1) ∧ queens(Xs).

safe(X,L,N) ⇔
   L = []
 ∨ L = [Y|Ys] ∧ noattack(X,Y,N) ∧ safe(X,Ys,N+1).
```

Anmerkung

- The predicate `make_domains` declares each of the row variables from `Qs` to range over the values `1` to `N`. These correspond to the row variables `X1,...,XN`. To set the variable `X` to have initial domain `L` we use the notation `X∈L`.
- The predicate `queens` ensures that no queen falls on the same row or diagonal as any other queen. It iterates through the list of queens calling `safe` to ensure that each queen `X` does not fall on the same row or diagonal as the remaining queens in the list.
- The predicate `safe` iterates through the queens in `Xs` adding `noattack` constraints to enforce that each queen in the list is not on the same row or diagonal as `X`. The implementation of the `noattack` constraint is given below.
- Finally the predicate `labeling` is called, to ensure that a valid solution is found. The predicate `labeling` iterates through each variable `X` in the list of variables to be labeled, calling `member(X,D)`, where `D` is the domain of `X`, to set `X` to each of the remaining values in its domain. The variables are tried in the order of their appearance in the list. For large n more sophisticated labeling strategies can be implemented.

n-Queens Constraints

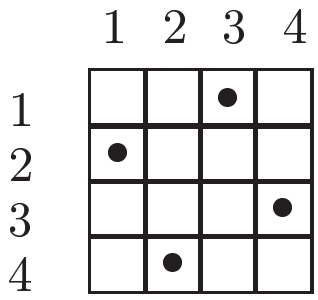
```
X∈[] ⇔ false.
noattack(X,Y,N) ∧ Y∈D ⇔
    remove_values([X,X+N,X-N],D,D1) ∧ Y∈D1.
noattack(Y,X,N) ∧ Y∈D ⇔
    remove_values([X,X+N,X-N],D,D1) ∧ Y∈D1.

labeling([]) ⇔ true.
labeling([X|Xs]) ∧ X∈L ⇔
    member(X,L) ∧ labeling(Xs).

member(X,[Y|Ys]) ⇔ X=Y ∨ member(X,Ys).
```

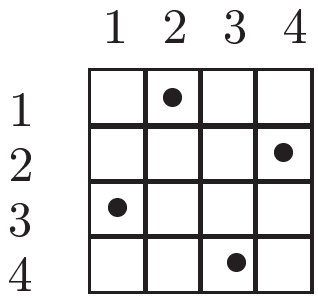
Anmerkung

To solve the N-queens problem, our constraint solver only needs to handle the `noattack` and $\in$ constraints. `noattack(X,Y,N)` is true if `Y`$\neq$`X` $\wedge$ `Y`$\neq$`X+N` $\wedge$ `Y`$\neq$`X-N` holds, i.e. `noattack(X,Y,N)` ensures that a queen `Y` is not on the same row or diagonal as a queen `X`.

The first rule ensures that the domain for `X` cannot be empty. The second and third rule remove the values `X, X+N` and `X-N` from the domain of `Y` provided `X` is a number.

n-Queens Solutions





Anmerkung

The program executes the goal `solve(4,[X1,X2,X3,X4])` as follows. After the domain declarations the domain is `X1`$\in$`[1,2,3,4]` $\wedge$ `X2`$\in$`[1,2,3,4]` $\wedge$ `X3`$\in$`[1,2,3,4]` $\wedge$ `X4`$\in$`[1,2` The `safe` predicate adds the `noattack` constraints: `noattack(X1` $\wedge$ `noattack(X1,X3,2)` $\wedge$ `...`$\wedge$ `noattack(X3,X4,1)`. As each of these constraints involves variables with no fixed value, no propagation occurs. In order to guarantee that a valid solution is found `labeling` is called. The first variable to be assigned is `X1`. Trying the first value in the initial domain, `1`, propagation using the rules from the constraint solving part reduces the domains of `X2,X3` and `X4`, i.e. `X2`$\in$`[3,4]` $\wedge$ `X3`$\in$`[2,4]` $\wedge$ `X4`$\in$`[2,3]`. Execution of `labeling` continues until a solution is found.

The program gives two solutions to the goal `solve(4,Qs)`, namely `Qs = [2,4,1,3]` and `Qs = [3,1,4,2]`.

n-Queens Code

```
solve(N,Qs) <=>
    make_domains(N,Qs), queens(Qs), labeling(Qs).

queens(L) <=> (L = []);
          (L = [X|R], safe(X,R,1),queens(R)).

safe(X,L,N) <=> (L = []);
          (L = [Y|R], noattack(X,Y,N), safe(X,R,N+1)).

X in [] <=> false.

noattack(X,Y,N),  Y in D <=> ground(X) |
   remove([X,X+N,X-N],D,D1),
   Y in D1.
noattack(Y,X,N),  Y in D <=> ground(X) |
   remove([X,X+N,X-N],D,D1),
   Y in D1.

labeling([]) <=> true.
labeling([X|Xs]), X in L <=> member(X,L), labeling(Xs).

member(X,[Y|Ys]) <=> X=Y ; member(X,Ys).
```

Anmerkung

Gaussian Variable Elimination

$a_1 * X_1 + \ldots + a_n * X_n = b$ is written as
`[a1*X1,..., an*Xn] = b`.
`delete(E,L1,L2)` holds if the removal of `E` from the list `L1` results in the list `L2`.

```
[A1*X|P1]=0, P=0 <=>
   delete(A2*X,P,P2)
   |
   compute(P2+P1*A2/A1,P3),
   [A1*X|P1]=0,
   P3=0.

P = B <=> P=[] | B=0.
```

Anmerkung

```
[A1*X|P1]=0, P=0 <=>
   kommt_vor_in(X,P) |
   entferne(A2*X,P,P2),
   berechne(P2+P1*A2/A1,P3),
   [A1*X|P1]=0,
   P3=0.       % P w/o X
```

E. Monfroy (M. Rusinowitch), Nancy, 1996:
GroAk: lineare (CHR) mit nichtlinearer Variablenelimination (Gröbner Basen).

Application: Finance

```
% D: Amount of Loan, Debt
% T: Duration of loan in months
% Z: Interest rate per month
% R: Monthly rate of payments
% S: Balance of debt after T months

mortgage(D, T, Z, R, S) <=>
        T = 0,
        D = S
          ;
        T > 0,
        T1 = T - 1,
        D1 = D + D*Z - R,
        mortgage(D1, T1, Z, R, S).
```

Example Mortgage

`mortgage(100000,360,0.01,1025,S)` yields `S=12625.90`.

`mortgage(D,360,0.01,1025,0)` yields `D=99648.79`.

`S=<0, mortgage(100000,T,0.01,1025,S)` yields `T=374, S=-807.96`.

`mortgage(D,360,0.01,R,0)` yields `R=0.0102861198*D`.

If the interest rate `Z` is unknown, the equation `D1 = D + D*Z - R` will be non-linear after one recursion step, since `D1`, the new `D`, is not determined either.

Conclusions

Constraint Programming

Generic Framework for
- modelling with incomplete information
- solving of combinatorial problems

in decision support systems.

Constraint Handling Rules

Declarative language for constraint programming.
- Executable specifications.
- Good theoretical properties.
- Implementations and libraries available.

Flexible and maintainable intelligent software.

Anmerkung

Constraint Programming

Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

[Eugene C. Freuder, *Constraints Journal*, 1997]

Anmerkung

60ties,70ties Constraint networks in artificial intelligence.
70ties Logic programming (Prolog).
80ties Constraint logic programming.
80ties Concurrent logic programming.
90ties Concurrent constraint programming.
90ties Commercial applications.

Constraint Handling Rules

For the theoretician meta-theorems can be proved and analysis techniques invented once and for all; for the implementor different constructs can be implemented once and for all; for the user only one set of ideas need to be understood, though with rich (but disciplined) variations (constraint systems).

[P. van Hentenryck, V.A. Saraswat, Strategic Directions in Constraint Programming, *ACM Computing Surveys*, 1996]

Anmerkung