CogSysIII Lecture 12: Automated Graph Layout

Human Computer Interaction

SS 2005

Ute Schmid (lecture)
Emanuel Kitzelmann (practice)

Applied Computer Science, Bamberg University

Aspects of Layout

- Determining sizes and positions of visual objects in a presentation
- Examples: graphical or textual UIs, WWW documents, newspapers
- Intersection of AI and HCI research
- Relevance: Amount a data to present overtakes ability to layout manually
- Layout has impact on quality of communication of information (is relationship between objects clear, positioning of relevant objects, etc.)

Aspects of Layout cont.

- Currently: mostly done by hand (graphic designers, layout experts)
- Automated layout: Use of algorithms for layout
- Simon Lok and Steven Feiner. A Survey of Automated Layout Techniques for Information Presentations. In Proceedings of SmartGraphics 2001

Techniques

- Simple techniques: layout manager (Java Swing, LaTeX, Word, ...)
- Constraint Satisfaction Techniques
- Learning Techniques
- Evaluation Techniques

Layout Manager

- Commands as “add button”, “add button to this part of window”, optionally specification of numerical constraints
- LM chooses positions and sizes at run time governed by a set of constraints (simple layout policy and parameters specified by programmer)
- Layout policies: strict horizontal/vertical layout, row-major or column-major (wrapping), border layout, grid layout
- Parameters: preferred/min/max width, heights, spacing
- Layout manager: no calculation of layout but instantiation of a layout

Java Container 1

- Komponenten müssen innerhalb eines `Container` gepackt werden.
- Ein `Container` ist eine `Component`, die andere `Components` enthalten kann.
- Alle speziellen Container sind Unterklassen von `awt.Container`.
- Typische Container: Fenster und Dialog-Boxen.
- Häufi g werden Container in andere Container verschachtelt.
- Achtung: `JFrame`, `JWindow` und `JDialog` können nur als “äussertster” Container verwendet werden!
- Manche Container stellen Information auf spezielle Weise dar, manche haben Restriktionen bzgl. Zahl und Art der Komponenten, die sie enthalten können, manche sind generisch (beliebig konfi gurierbar).

Java Container 2

- Mit der `Window`-Methode `pack()` wird ein Fenster ge-rendert (*rendering*), also in eine graphische Darstellung gebracht. Dabei wird die Größe des Fensters – in Abhängigkeit vom Layout der Unter-Komponenten – ermittelt.
- Jedes Fenster ist mit einem *LayoutManager* (ein Interface in `java.awt`) verbunden. Dieser wird mit `pack()` aktiviert.
- Typische Layout-Manager sind `BorderLayout` (Unterteilung eines Containers in fünf Bereiche, siehe unten) oder `ScrollPaneLayout` (Default für `JScrollPane`).
- Typische Schritte zum Erzeugen einer GUI:
  1. Erzeugung der Container
  2. Erzeugen der Components
  3. Hinzufügen der Components zum Container $\hookrightarrow$ `add()`

Java Container 3

```
JFrame frame = new JFrame("HelloWorldSwing");   // Container
final JLabel label = new JLabel("Hello World"); // Component
frame.getContentPane().add(label);
```

Java Container 4

- Spezielles Verhalten der top-level Container `JFrame`, `JWindow`, `JDialog`:
  - Wenn solche Container erzeugt werden, erzeugen diese automatisch eine Unterklasse `JRootPane`, die eine Unterklasse von `JComponent` ist und das Interface `RootPaneContainer` implementiert.
  - Alle Komponenten werden in `JRootPane` eingefügt. Die Methode `getContentPane()` liefert denjenigen Container, in den die Komponenten eingefügt werden sollen.

Java Layout Management

- `JFrame` und `JDialog` sind generische Komponenten. Sie benutzen `JPanel` als *default content pane* und spezifi zieren kein vordefi niertes Layout der Komponenten.
- Hier muss ein `LayoutManager` defi niert werden, um die Komponenten im Container anzuordnen.
- Default-Layout ist vorgegeben (d.h. entsprechendes Objekt existiert). Z.B. für `JFrame`: `BorderLayout`.
- `BorderLayout` erlaubt bis zu 5 Komponenten mit den Positionen North, South, East, West, Center.

Border Layout

Java Example

```
import java.awt.*;
import javax.swing.*;

public class BorderWindow extends JFrame {

    public BorderWindow() {
        Container contentPane = getContentPane();
        //Use the content pane's default BorderLayout.
        //contentPane.setLayout(new BorderLayout()); //unnecessary

        contentPane.add(new JButton("Button 1 (NORTH)"),
                        BorderLayout.NORTH);
        contentPane.add(new JButton("2 (CENTER)"),
                        BorderLayout.CENTER);
        contentPane.add(new JButton("Button 3 (WEST)"),
                        BorderLayout.WEST);
        contentPane.add(new JButton("Long-Named Button 4 (SOUTH)"),
                        BorderLayout.SOUTH);
        contentPane.add(new JButton("Button 5 (EAST)"),
                        BorderLayout.EAST);
    }
```

Java Example

```
    public static void main(String args[]) {
        BorderWindow window = new BorderWindow();
        window.setTitle("BorderLayout");
        window.pack();
        window.setVisible(true);
    }
}
```

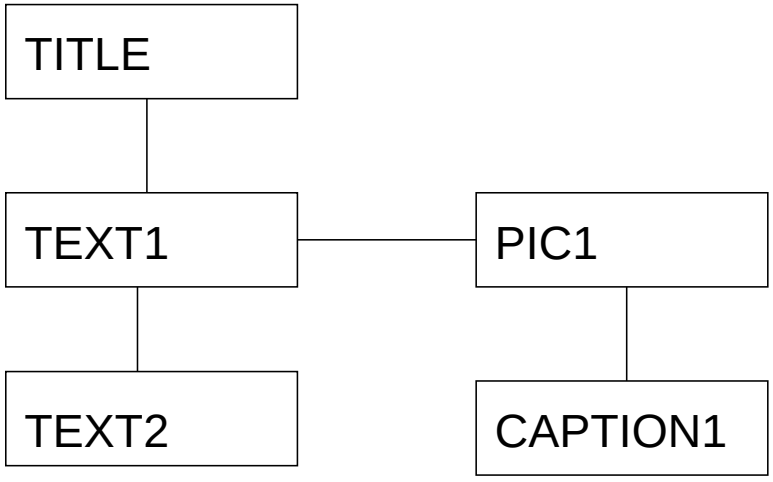
Java Beans

- Bean: Wiederverwendbare Software-Komponente, die in einem Builder-Tool visuell manipuliert werden kann. Beispiel: BDK (Java Beans Development Kit).
- Typisch für graphische Benutzeroberflächen.
- Schreiben von Beans: z.B. neue graphische Komponenten
  sollten über Properties konfigurierbar sein und entsprechende `get-` und `set`-Methoden anbieten.
- Nutzen von Beans: Zusammenstecken und Konfigurieren von Komponenten und mit Code verbinden.

Constraint-Satisfaction

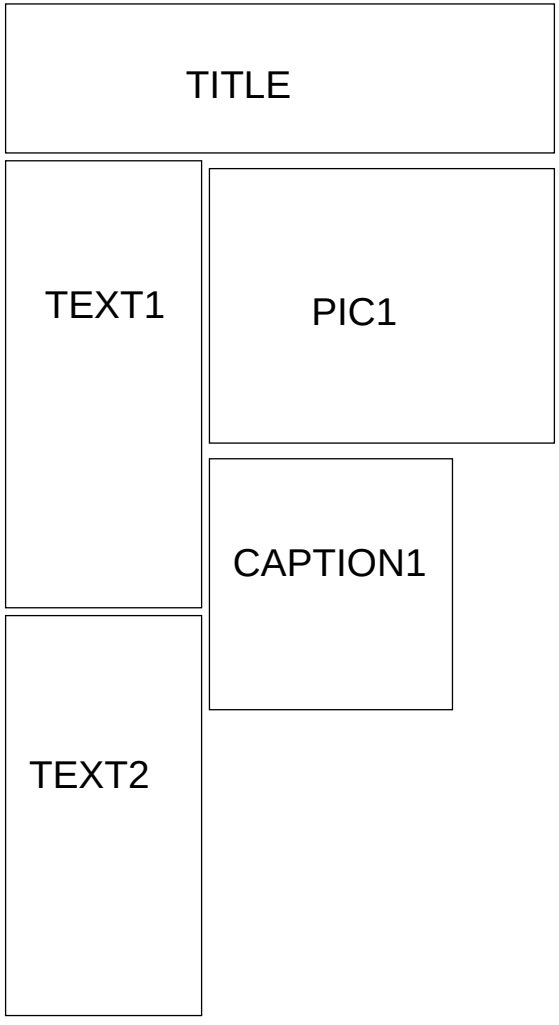
- Majority of research in automated layout
- see e.g., lecture of Ch. Schlieder (K-Infl)
- Types of Constraints
  - Abstract: high level (“TEXT1 references PIC1”)
  - Spatial: “CAPTION1 below PIC1”
- Generating spatial cs from abstract cs is the most challenging part of automated layout

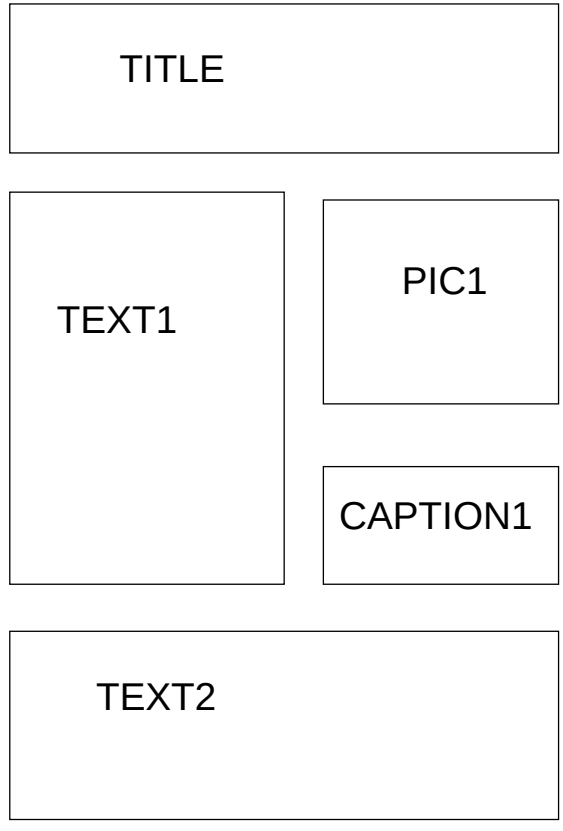
Example



Spatial Constraints:	Abstract Constraints
TITLE above TEXT1	TITLE describes TEXT1
TITLE fullpagewidth	TITLE isimportant
TEXT1 leftof PIC1	TEXT1 references PIC1
CAPTION1 below PIC1	CAPTION1 describes PIC1
TEXT2 below TEXT1	TEXT2 follows TEXT1

A Legal and a Helpful Layout





Constraint Satisfaction Problems

- Multidimensional selection problems:
  - Given a set of variables, each with a set of possible values (a domain) and a set of constraints (relations over the variables)
  - assign a value to each variable such that the constraints are satisfied.
- Example: Eight Queens Problem
  find an assignment of board positions for the queens such that no queen can beat another one.

CSP and Problem Solving

- (Relation to KogSysI)
- Again: use search algorithm
- In contrast to problem solving and game playing,
  - There is no predefined starting node
  - only the solution, not the path to reach the goal, is important.
  - Goal is a set of conditions (constraints) which can be fulfilled not at all, partially or completly

Hard and Soft Constraints

- Hard constraints: satisfying the set of all constraints given
  $\hookrightarrow$ satisfiability problems
- Soft constraints: some violation is allowed, minimize some cost function (e.g., quadratic penalty)
  $\hookrightarrow$ optimization problems
- In the following, we only consider hard constraints

Further restrictions

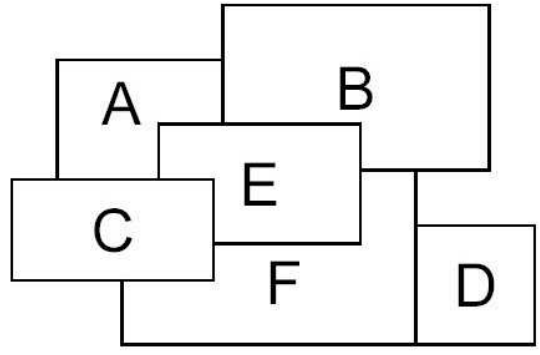
- finite, discrete domains
- only unary and binary constraints

Examples

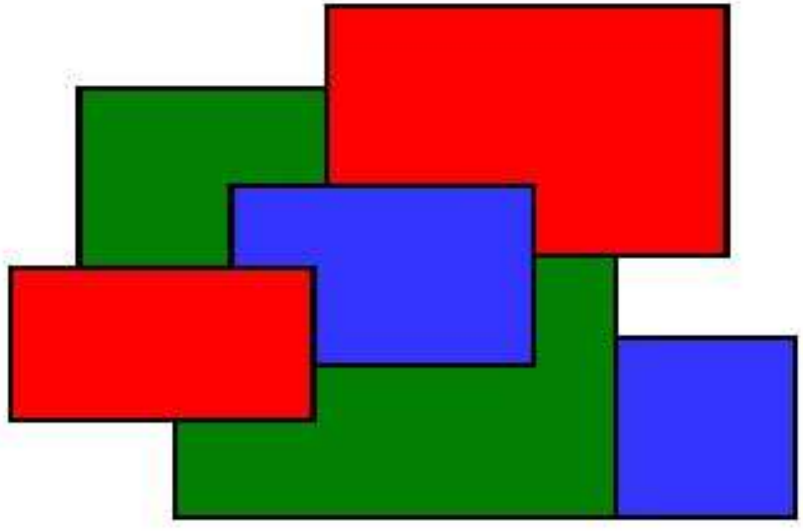
- Map coloring: For a given set of k colors (e.g. 4), find an assignment of colors to each area (e.g. country) such that no neighboring regions have the same color.
- Scheduling problems (e.g., job scheduling): assignment of start and end times to a set of tasks, such that time constraints (e.g. delivery deadlines) and resource constraints (e.g. usage of certain machines) are satisfied.
- Model checking
- Language processing
- Configuration/Layout
- Crossword puzzles
- Solving equations/in-equations
- Crypt-arithmetic
- Interpretation of visual scenes
- ...

Map Coloring

- Problem: Color a card with three colors such that adjacent regions have different colors
- Constraint Variables: {A, B, C, D, E, F}
- Constraint Values: {red, green, blue}

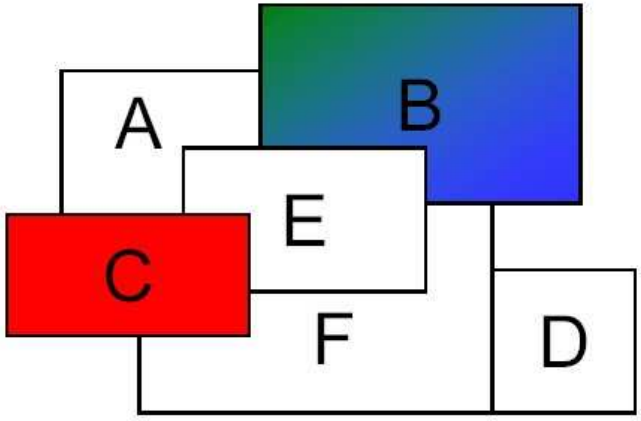


Map Coloring, A Solution



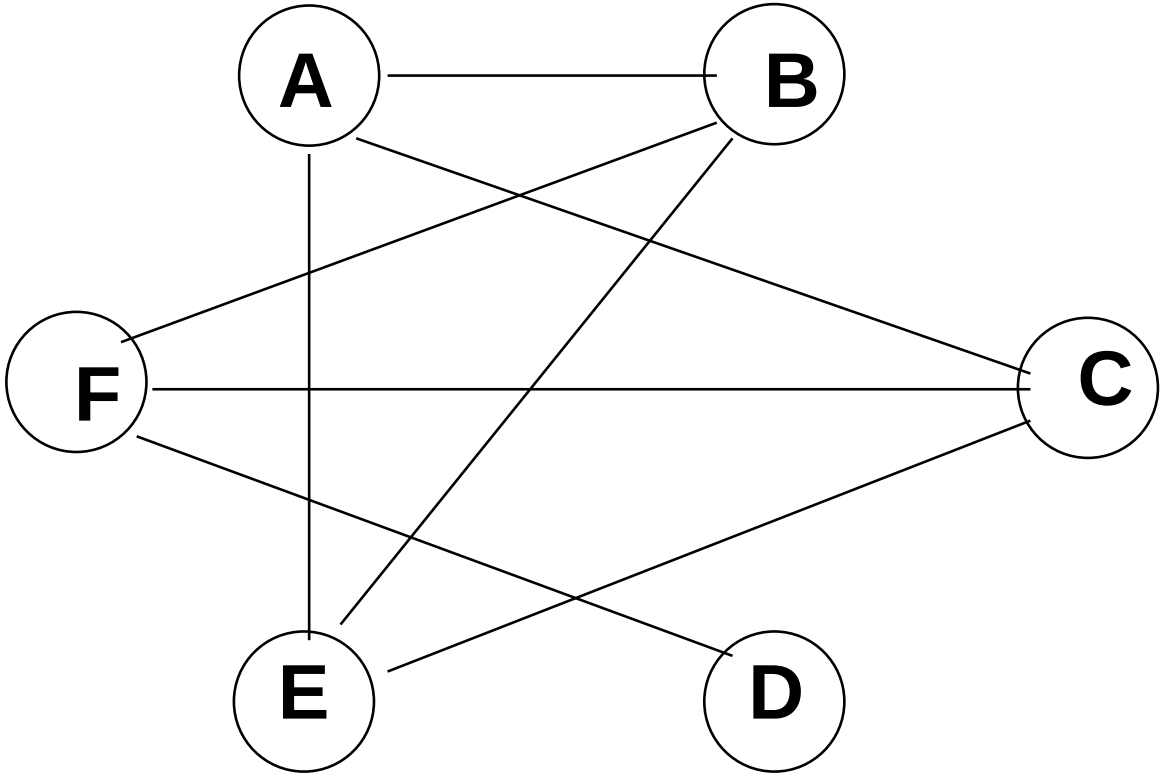
Definition Constraint

- Set of admissible assignements of values to variables
- unary: Restriction of assignments to a single variable
  e.g., $B \in$ {green, blue}, $C \in$ {red, green},
- binary: Restriction of assignements for pairs of variables
  e.g., $(A, B) \in$ {(green, blue), (blue, green), (green, red), (red, green), (blue, red), (red, blue)}



Constraint-Graph

- or constraint net
- Nodes represent variables
- Arcs represent binary constraints



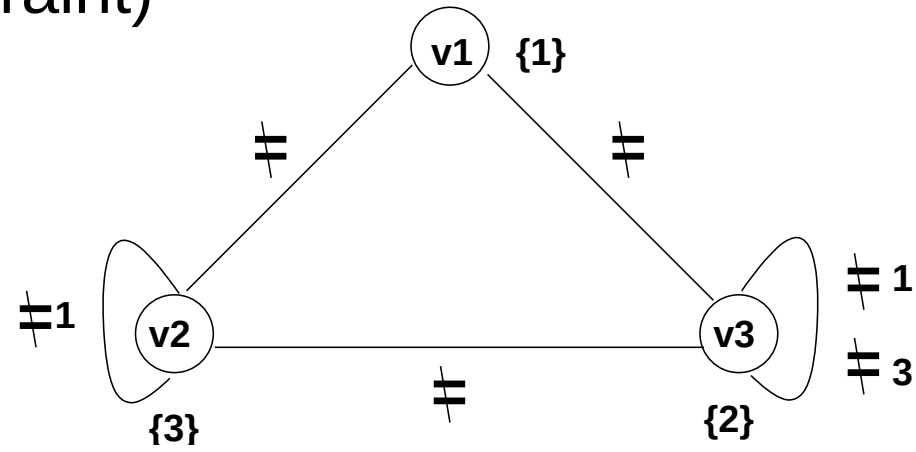
Definition of CSP

- A constraint satisfaction problem (CSP) is characterized by
  - a set V of variables $\{v_1, \ldots, v_n\}$,
  - the possible values (domains) D_i of variables where $D = D_1 \times \ldots \times D_n$ is the assignment space,
  - a set of constraints, i.e., relations $R_k \subseteq D_{k_i} \times \ldots \times D_{k_j}$ for some domains.
- Task: Find one or all tuples of $D_1 \times \ldots \times D_n$ such that all constraints hold.

A constraint network is a labeled graph where the nodes represent the variables and the edges represent the relations between them.

Simple Example

- $V = \{v_1, v_2, v_3\}$ with $D_i = \{1, 2, 3\}$ for $i = 1 \ldots 3$.
- $P_1 : v_2 \neq 1$ (unary constraint)
- $P_2 : v_3 \neq 1$
- $P_3 : v_3 \neq 3$
- $P_4 : v_1 \neq v_2$ (binary constraint)
- $P_5 : v_1 \neq v_3$
- $P_6 : v_2 \neq v_3$

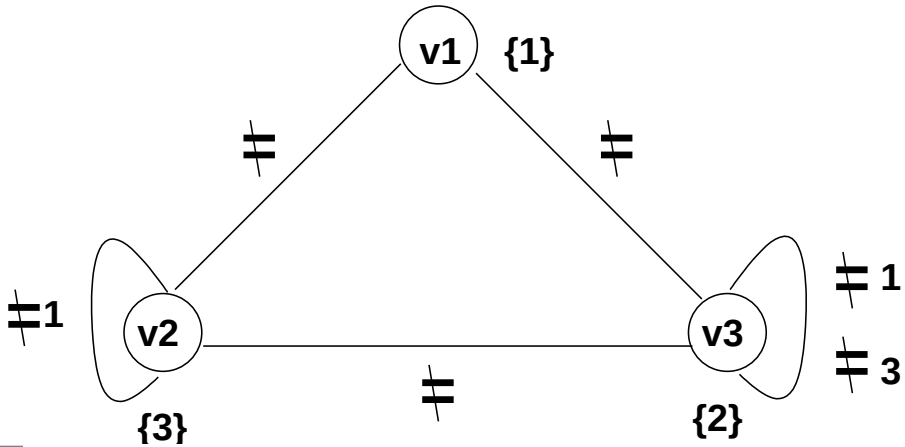


Algorithms for CSP

- Generate and test: because of problem complexity in general not feasible
- Backtracking algorithms: (many variants, e.g. hill climbing)
  - Start with the empty assignment
  - Systematically explore D by instantiating the variables in some order:
    - Evaluate each constraint as soon as all its variables are bound.
    - Any partial assignment that does not satisfy the constraint can be pruned.
- Genetic algorithms
- Consistency Algorithms

Consistency Algorithms

- Again, many variants.
  General idea: Prune domains as much as possible before selecting values from them.
  - Example: $D_3 = \{1, 2, 3\}$ is not domain consistent because $v_3 = 1$ and $v_3 = 3$ violates constraints P_2 and P_3.
- Consistency algorithms are local strategies, based on local consistencies.



Node Consistency

- In the following we write P_i to represent the unary constraints for variable/node v_i, and
- P_{ij} to represent the binary constraints for variables/nodes v_i and v_j.
- We assume that all relations are symmetrical:
  $\forall x, y \; P_{ij}(x, y) \leftrightarrow P_{ji}(y, x)$
  (for each relation, the inverse can be constructed).

- Node consistency
  - The domain D_i for a variable/node v_i contains only values for which the unary constraints $P_i(v_i)$ hold:
    $\forall x \; x \in D_i \rightarrow P_i(x)$
  - A node can be made consistent by performing a domain restriction operation: $D_i \leftarrow D_i \cap \{x \mid P_i(x)\}$

Arc Consistency

- Arc consistency (2-consistency)
  - The domains D_i and D_j contain only values for which the binary constraints $P_{ij}(v_i, v_j)$ hold:
    $\forall x\ P_i(x) \rightarrow \exists y\ D_j(y) \wedge P_{ij}(x, y)$
  - An edge between two nodes can be made consistent by performing a arc consistency domain restriction operation, removing all elements from D_i that have no corresponding element in D_j:
    $D_i \leftarrow D_i \cap \{x \mid \exists y\ y \in D_j \wedge P_{ij}(x, y)\}$

Arc Consistency and Solution

- If all arcs are consistent:
  - if at least one domain is empty: no (globally consistent) solution
  - if each domain has a single value: unique solution
  - if some domains have more than one value: may or may not be a solution

Path Consistency

- Path consistency (3-consistency)
  - For all nodes v_i, v_j, v_k holds:
    $\forall x\ y\ P_{ij}(x,y) \rightarrow \exists z\ P_k(z) \wedge P_{ik}(x,z) \wedge P_{kj}(z,y)$
  - Path consistency can be achieved by iteratively performing the the following restriction operation until there are no more changes:
    $P_{ij} \leftarrow P_{ij} \cap \{(x,y) \mid \exists z\ P_{ik}(x,z) \wedge P_{kj}(z,y)\}$
- A further generalization is k-consistency (for each set of $k-1$ consistently instantiated variables exists a consistent instantiation of a k-th variable).

Remarks on Local Consistency

- Arc and path consistency algorithms are guaranteed to terminate for finite domains and generate the same set of solutions independently of the sequence in which variables are instantiated (many paths are equivalent, it suffices to follow some of them).
- Domain restriction operations do not change the solution set. Violated constraints cannot be corrected by further domain restriction operations (simple cutoff criterion).
- Local consistency algorithms might or might not generate globally consistent solutions!

Remarks on Local Consistency cont.

- Consistency algorithms rely on constraint propagation: If some domain is changed by applying a restriction operation, for all connected nodes it is checked whether this change affects their value sets, etc.
- A special class of consistency algorithms are relaxation algorithms where the set of values for each variable is stepwise reduced.
  (Example: Waltz Algorithm for labeling of visual scenes.)

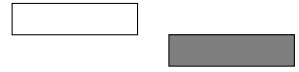
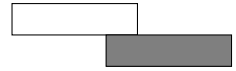
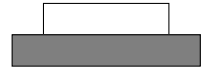
Example: Allen Calculus

- James F. Allen, 1983, Maintaining Knowledge about temporal intervals, *Communications of the ACM*, *26*(*1*), 832-843.
  Hans Guesgen, 1989, *Spatial reasoning based on Allen's temporal logic*. TR-89-049, ICSI, Berkleley, CA.
- $1D$, interval-based calculus
- Each interval X is an ordered pair of end points s_X and e_X.
- Interpretation $\mathcal{I}$: Mapping to real numbers with $s_X < e_X$.
- 13 base relations: pairwise disjoint and exhaustiv.

"*The palm tree is left of the computer.*" $P \prec C$

$s_P < s_C$, $s_P < e_C$, $e_P < s_C$, $e_P < e_C$

Allen's Base Relations

Symbol	Verbal Desc.	Illu	Point-Order
$X \prec Y$	X lies to the left of Y		$s_X < e_X < s_Y < e_Y$
$X\ m\ Y$	X touches Y at the left		$s_X < e_X = s_Y < e_Y$
$X\ o\ Y$	X overlaps Y from the left		$s_X < s_Y < e_X < e_Y$
$X\ s\ Y$	X lies left-justifi ed in Y		$s_Y = s_X < e_X < e_Y$
$X\ d\ Y$	X is completely in Y		$s_Y < s_X < e_X < e_Y$
$X\ f\ Y$	X lies right-justifi ed in Y		$s_Y < s_X < e_X = e_Y$
$X = Y$	X is equal Y		$s_X = s_Y < e_Y = e_X$

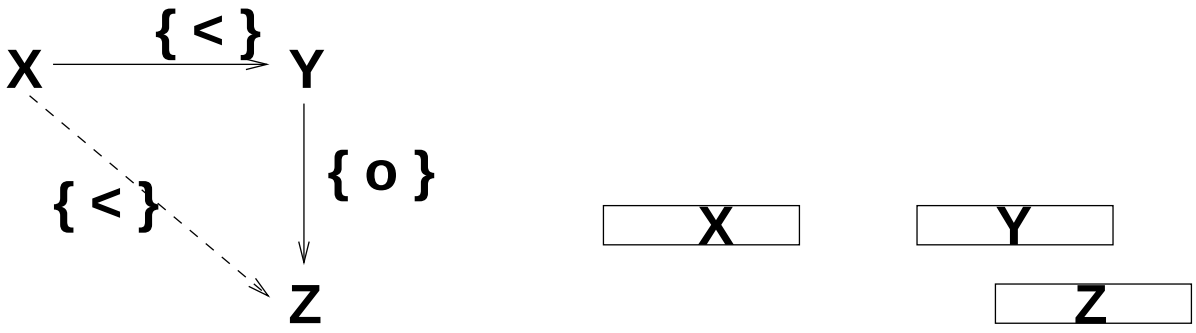
and the inverse relations (fi , di, si, oi, mi, $\succ$)

- Remark: symbol names are from temporal domain (before, meets, overlaps, starts, during, fi nishes)

The Inference Problem

"The TV is left of the table and the table overlaps the bench from the left."

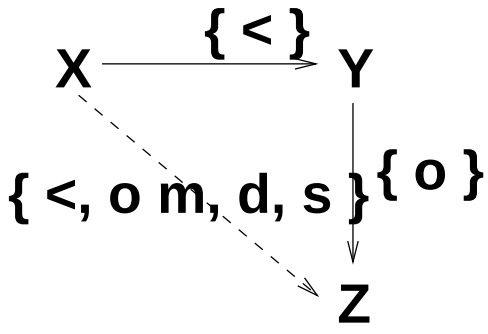
$(X \prec Y) \wedge (Y \; o \; Z)$

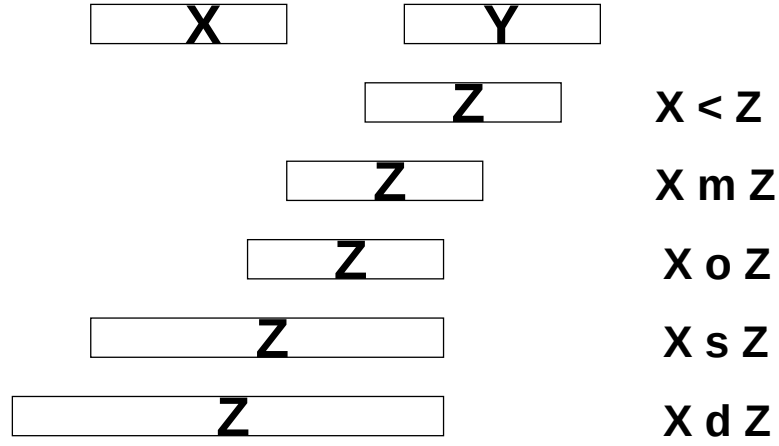


- Composition of relations: $I(R \circ S)J \Leftrightarrow \exists K : (I \; R \; K \wedge K \; S \; J)$.
- What does hold for $X(\prec \circ o)Z$?
- Inferable using the endpoint relations.
- Faster: pre-calculation and storage in a composition table
- Problem: Composition does not always result in a unique relation.

The Inference Problem cont.

$$(X \prec Y) \wedge (Y \; oi \; Z)$$





Composition Table

The composition table for the Allen Calculus is rather complex. To illustrate the idea, here is a composition table for precedence relations only.

	$<$	$=$	$>$
$<$	$<$	$<$	$<, =, >$
$=$	$<$	$=$	$>$
$>$	$<, =, >$	$>$	$>$

Allen's Interval Calculus

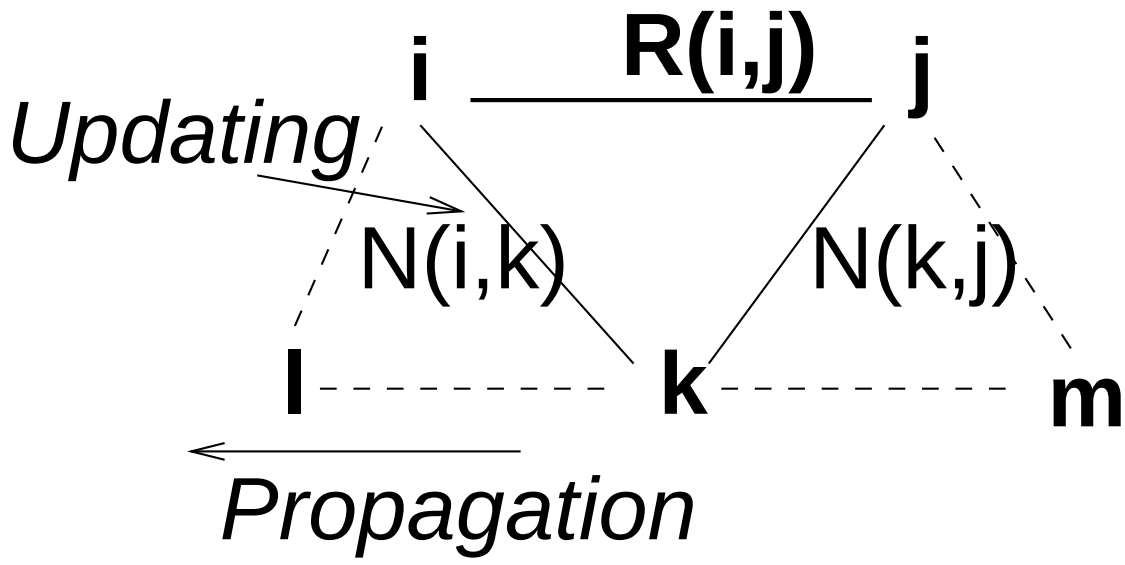
- Construct a network (graph) with intervals (1D objects) as nodes and relations as edges.
- To hold the complete information about relations between objects: if a new relation is inserted, immediately calculate all inferences and introduce the edges (transitive closure).

Operations over relations:

Inverse: $I \quad R^{-1}\ J \Leftrightarrow J\ R\ I$.
Intersection: $I\ (R \cap S)\ J \Leftrightarrow I\ R\ J \wedge I\ S\ J$.
Composition: $I(R \circ S)J \Leftrightarrow \exists K : (I\ R\ K \wedge K\ S\ J)$.

- Composition: introduced above
- Inverse: to follow a path it might be necessary to change edge-direction
- Intersection: Introducing new information to an already labelled edge.

Constraint Satisfaction Algorithm



- Insert new relation $R(i, j)$
- For all nodes k which are directly connected with i/j: Calculate constraints from k to j via i and from i to k via j.
- If there are changes: deal with $R(k, j)$/$R(i, k)$ in the same way.

Algorithmus cont.

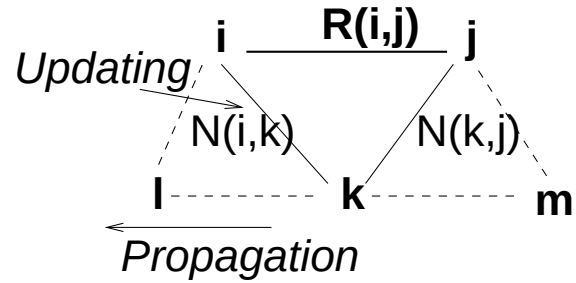
To **Add** R(i,j)

- Add $\langle i, j \rangle$ to queue *ToDo*;
- **While** *ToDo* is not empty **do**
  begin
  - Get next $\langle i, j \rangle$ from *ToDo*;
  - $N(i, j) \leftarrow R(i, j)$;
  - **For each** node k **do**
    begin
    - $R(k, j) \leftarrow N(k, j) \cap \mathbf{Constraints(N(k, i), R(i, j))}$
    - **If** $R(k, i) \subset N(k, i)$ **then** add $\langle k, i \rangle$ to *ToDo*;

    end
  - **For each** node k **do**
    begin
    - $R(i, k) \leftarrow N(i, k) \cap \mathbf{Constraints(R(i, j), N(j, k))}$
    - **If** $R(i, k) \subset N(i, k)$ **then** add $\langle i, k \rangle$ to *ToDo*;

    end

  end



Constraints(R_1, R_2)

- $C \leftarrow \epsilon$;
- **For each** r_1 in R_1
  - **For each** r_2 in R_2
    $C \leftarrow C \cup T(r_1, r_2)$;
- **Return** C;

(T: composition table)

Satisfiability and Inference

- There are 2^{13} possible interval relations, including the empty (contradiction) and the universal (no information) relation.
- Satisfi ability of a relation $X\ R\ Y$: there exists an interpretation of intervals in the domain of real numbers which satisfi es the endpoint relations.
- Satisfi ability of an interval formula $X\{R_1, \ldots, R_n\}$
  $\exists R_i \in \{R_1, \ldots, R_n\}$ with $X\ R_i\ Y$ is satisfi able.
- Satisfi ability of a fi nite set of interval formula ϕ: There exists an interpretation satisfying every fomula in ϕ (model).
  $\hookrightarrow$ Does there exist a globally consistent arrangement of intervals?
- A formula ϕ follows from a set of interval formula Θ ($\Theta \models \phi$), if it is satisfi ed in each model of Θ.
  $\hookrightarrow$ For each pair of intervals, fi nd the stronges relation R with $\Theta \models X\ R\ Y$.

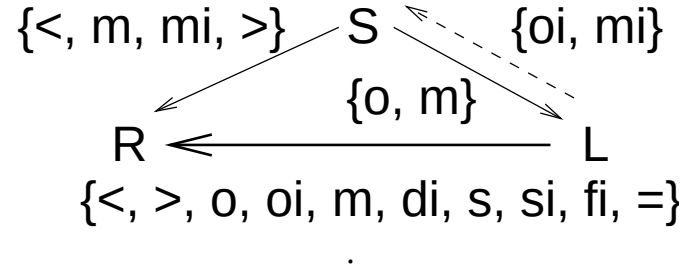
Example

- (1) $S\ \{o, m\}\ L$ (= $L\ \{oi, mi\}\ S$)
- (2) $S\ \{\prec, m, mi, \succ\} R$
- Calc **Constraints** ($\{o, m\}, \{\prec, m, mi, \succ\}$)

  Relation of L and R via S (converse: $L \rightarrow S$)

  - $T(oi, \prec) = \{\prec, o, m, di, fi\}$
  - $T(oi, m) = \{o, di, fi\}$
  - $T(oi, mi) = \{\succ\}$
  - $T(oi, \succ) = \{\succ\}$
  - $T(mi, \prec) = \{\prec, o, m, di, fi\}$
  - $T(mi, m) = \{s, si, =\}$
  - $T(mi, mi) = \{\succ\}$
  - $T(mi, \succ) = \{\succ\}$

  $\hookrightarrow L\ \{\prec, \succ, o, m, di, s, si, fi, =\}\ R$

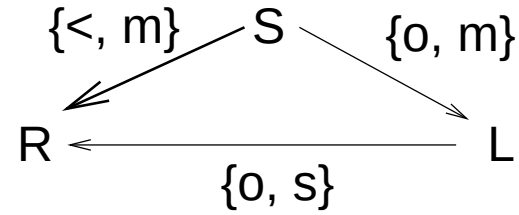


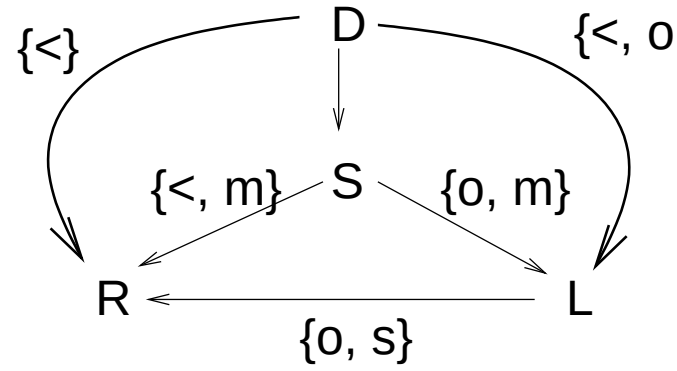
Example cont.

- (3) $L\ \{o, s, d\}\ R$
- Intersection with given relation: $L\ \{o, s\}\ R$
- Insert new constraint $\hookrightarrow$ Propagation: new constraint for S and R

  $S\ \{o, m\}\ L\ \{o, s\}\ R$
  - $T(o, o) = \{\prec, o, m\}$
  - $T(o, s) = \{o\}$
  - $T(m, o) = \{\prec\}$
  - $T(m, s) = \{m\}$

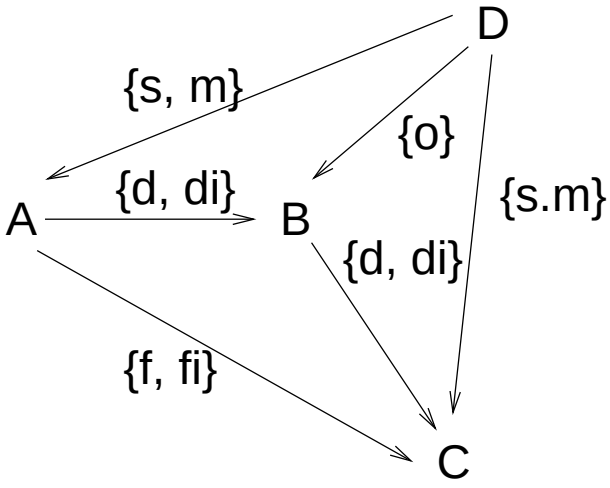
  $\hookrightarrow S\ \{\prec, o, m\}R$
- Intersection with $S\ \{\prec, m, mi, \succ\}\ R$: $S\ \{\prec, m\}\ R$
- (4) $D\ \{d\}\ S$
- New relations:
  - $D\ \{\prec\}\ R$
  - $D\ \{\prec, o, m, d, s\}\ L$

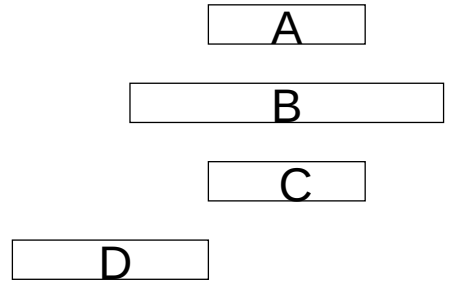




Remarks about the Algorithm

- local method: path consistency (quadratic effort)
- problem: although the algorithm cannot generate inconsistencies, it might not detect (global) inconsistencies.
- solution: do not only check triples of nodes but also quadruples, quintuples, etc. (exponential effort!)





A d B

B di C

D m A
D o B

A f C –> nicht D m A
A fi C –> nicht D m A

Cognitive Adequacy of the Allen Calculus

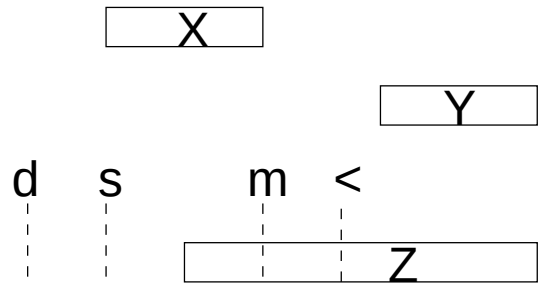
- Human-Machine-Interaction:
  - human understanding of the interval relations and their names
    $\hookrightarrow$ Psychological studies of **conceptual adequacy** (Knauff, 1997)
    Generate spatial relations, ratings of acceptance, recall/recognition experiments (discernability of basic relations)
  - human reasoning with interval relations
    $\hookrightarrow$ Psychological studies of **inferential adequacy** (Knauff, Rauh, Schlieder, Strube, 1998)

$\hookrightarrow$ Output of the algorithm should be inaccord with human expectations

Setting of an Experiment

- Task: for *3-term-series* ($X\ R\ Y$ und $Y\ R'\ Z$)
  name *one* possible conclusion ($X\ R''\ Z$).
  $\hookrightarrow$ A significant number of subjects names the same conclusion.
  Preferred mental model!

  $X \prec Y, Y\ f\ Z \rightarrow X\ o\ Z\ (\{\prec, m, o, s, d\})$



- Simulation-model (Schlieder, 1995)
- Sequence effect of premises.