Analysis and Verification

LVA 185.276, VU 2.0, ECTS 3.0
SS 2019

(Latest Update: 5 June 2019)

Jens Knoop

Technische Universität Wien
Information Systems Engineering
Compilers and Languages

Table of Contents
(Selected Chapters)

Table of Contents (1)

Part III: Analysis

Table of Contents (2)

Table of Contents (3)

Appendices

Table of Contents (4)

Table of Contents (5)

Table of Contents (6)

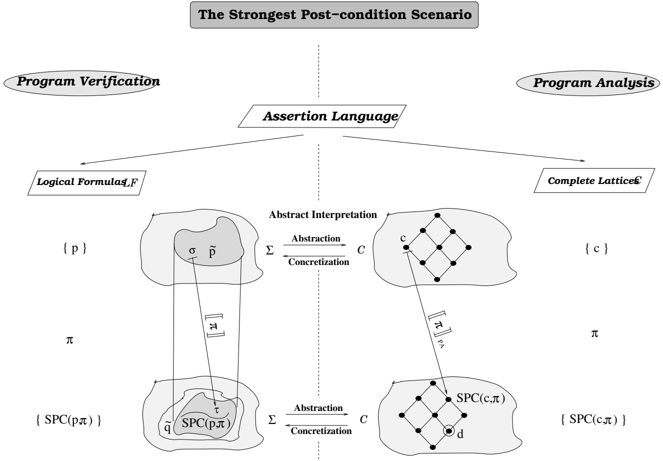
Part III

Analysis

Chapter 7

Data Flow Analysis

Motivation: Verification vs. Data Flow Analysis



SPC(p,π) $\in$ *LF* *must satisfy:*

(1) $\models_{PV} \{\, p \,\}\pi\{\, SPC(p,\pi) \,\}$

(2) $\forall\, q \in LF.\ \models_{PV} \{\, p \,\}\pi\{\, q \,\}$ implies $SPC(p,\pi) \Longrightarrow q$

SPC(c,π) $\in$ *C* *must satisfy:*

(1) $\models_{PA} \{\, c \,\}\pi\{\, SPC(c,\pi) \,\}$

(2) $\forall\, d \in C.\ \models_{PA} \{\, c \,\}\pi\{\, d \,\}$ implies $SPC(c,\pi) \sqsupseteq d$

Chapter 7.1

Preliminaries

Chapter 7.1.1

Flow Graphs

Flow Graphs

...as representations of WHILE programs.

Definition 7.1.1.1 (Flow Graph)

A flow graph is a quadruple $G = (N, E, \mathbf{s}, \mathbf{e})$ with

- N, set of nodes.
- $E \subseteq N \times N$, set of edges.
- $\mathbf{s}$, distinguished start node w/out any predecessors.
- $\mathbf{e}$, distinguished end node w/out any successors.

Nodes represent program points, edges the branching structure of G. Every node of G is assumed to lie on a path from $\mathbf{s}$ to $\mathbf{e}$.

Node-labelled vs. Edge-labelled Flow Graphs

Given a flow graph, instructions (i.e., assignments, tests) can be represented by

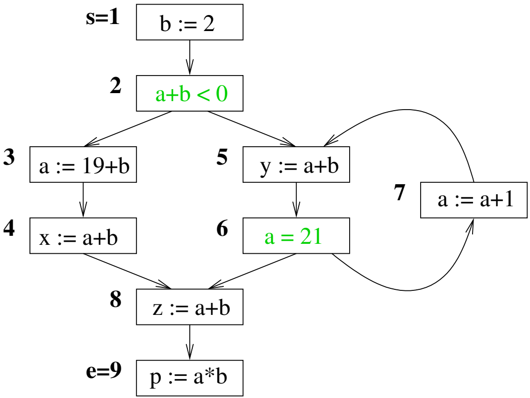
- nodes
- edges

leading to

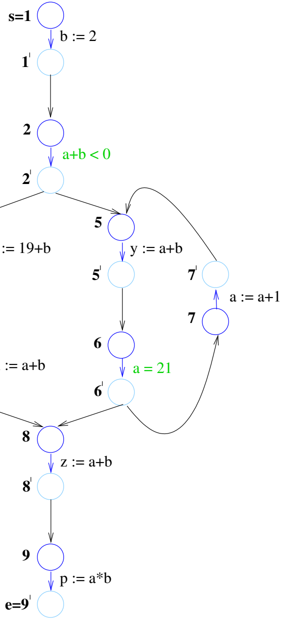
- node-labelled
- edge-labelled

flow graphs, respectively.

Example: A Node-Labelled Flow Graph

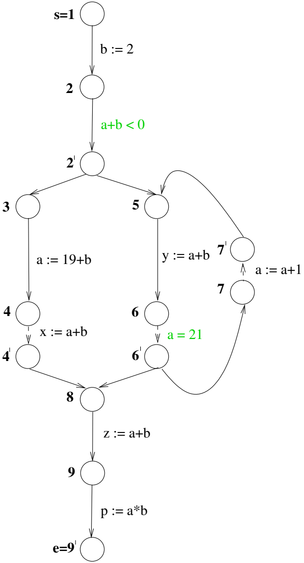


Example: An Edge-Labelled Flow Graph

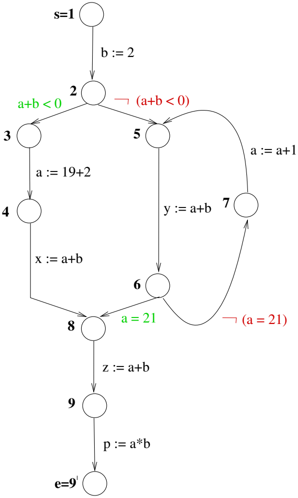


Edge-Labelled Flow Graph after Cleaning Up

a)



b)



Predecessor Nodes, Successor Nodes, Paths

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, m, n be two nodes of N.

Definition 7.1.1.2 (Predecessor, Successor Nodes)

- $pred_G(n) =_{df} \{ m \mid (m, n) \in E \}$ denotes the set of predecessor nodes of n.
- $succ_G(n) =_{df} \{ m \mid (n, m) \in E \}$ denotes the set of successor nodes of n.

Definition 7.1.1.3 (Paths)

- A sequence of edges $\langle (n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k) \rangle$ with $m_i = n_{i+1}$, $1 \leq i < k$ is called a path from n_1 to m_k.
- $\mathbf{P}_G[m, n]$ denotes the set of all paths from m to n.

Note, if G is obvious from the context, we drop index G and write $pred$, $succ$, and $\mathbf{P}$ instead of $pred_G$, $succ_G$, and $\mathbf{P}_G$, resp.

In the following

...we consider

- edge-labelled

flow graphs, which are pragmatically advantageous by requiring less notational overhead. Moreover, we do not evaluate tests in order to avoid (some) undecidabilities, leading us to so-called non-deterministic flow graphs.

Note, advantages and disadvantages of particular flow graph variants as program representations are discussed in

- Appendix B: Pragmatics of Flow Graph Representations

Chapter 7.1.2

Complete Lattices

Partially Ordered Sets, Complete Lattices

Definition 7.1.2.1 (Partially Ordered Set)

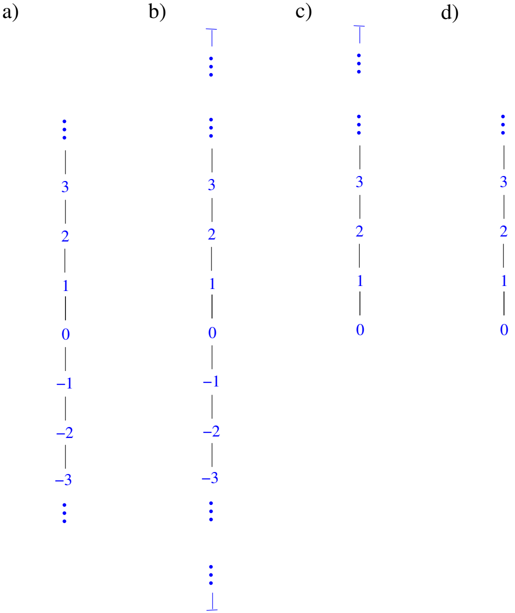
Let S be a set and $R \subseteq S \times S$ be a relation on S. Then (S, R) is called a partially ordered set (dtsch. partiell geordnete Menge) iff R is reflexive, transitive, and anti-symmetric.

Definition 7.1.2.2 (Lattice, Complete Lattice)

Let $(P, \sqsubseteq)$ be a partially ordered set. Then $(P, \sqsubseteq)$ is a

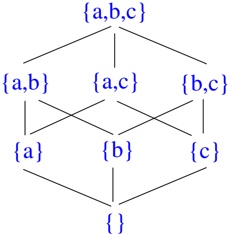
- lattice (dtsch. Verband), if every finite nonempty subset P' of P has a least upper bound and a greatest lower bound in P.
- complete lattice (dtsch. vollständiger Verband), if every subset P' of P has a least upper bound and a greatest lower bound in P.

Examples: Partially Ordered Sets and Lattices



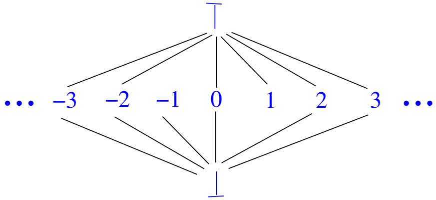
Examples: Complete Lattices

a)

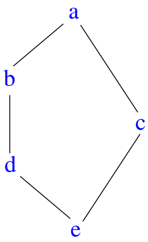


b)

c)



d)



Notions, Notations for Lattices

Let $(C, \sqsubseteq)$ be a complete lattice, $C' \subseteq C$ a subset of C. Then

- $\bigsqcap C'$ denotes the greatest lower bound of C'.
- $\bigsqcup C'$ denotes the least upper bound of C'.
- $\bigsqcap C = \bigsqcup \emptyset$ is the least element of C, denoted by $\bot$.
- $\bigsqcup C = \bigsqcap \emptyset$ is the greatest element of C, denoted by $\top$.

This gives rise to write a complete lattice as a six-tuple

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$

where $\sqcap$, $\sqcup$, $\bot$, and $\top$ are read as meet, join, bottom, and top, respectively.

Descending, Ascending Chain Condition

Definition 7.1.2.3 (Chain Condition)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a lattice. $\widehat{C}$ satisfies the

1. descending chain condition (dtsch. absteigende Kettenbedingung), if every descending chain gets stationary, i.e., for every chain $c_1 \sqsupseteq c_2 \sqsupseteq \ldots \sqsupseteq c_n \sqsupseteq \ldots$ there is an index $m \geq 1$ with $c_m = c_{m+j}$ for all $j \in \mathbb{N}$.
2. ascending chain condition (dtsch.aufsteigende Kettenbedingung), if every ascending chain gets stationary, i.e., for every chain $c_1 \sqsubseteq c_2 \sqsubseteq \ldots \sqsubseteq c_n \sqsubseteq \ldots$ there is an index $m \geq 1$ with $c_m = c_{m+j}$ for all $j \in \mathbb{N}$.

Monotonicity, Distributivity, and Additivity

...are important properties of functions on lattices:

Definition 7.1.2.4 (Monotonicity)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : \mathcal{C} \to \mathcal{C}$ be a function on $\mathcal{C}$. Then f is called

- monotonic iff $\forall\, c, c' \in \mathcal{C}.\ c \sqsubseteq c' \Rightarrow f(c) \sqsubseteq f(c')$
  (Preservation of the order of elements)

Definition 7.1.2.5 (Distributivity, Additivity)

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : \mathcal{C} \to \mathcal{C}$ be a function on $\mathcal{C}$. Then f is called

- distributive iff $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcap C') = \bigsqcap \{ f(c) \,|\, c \in C' \}$
  (Preservation of greatest lower bounds)
- additive iff $\forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcup C') = \bigsqcup \{ f(c) \,|\, c \in C' \}$
  (Preservation of least upper bounds)

Monotonicity

...characterized in terms of the preservation of greatest lower and least upper bounds:

Lemma 7.1.2.6

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice, $f : \mathcal{C} \to \mathcal{C}$ a function on $\mathcal{C}$. Then:

$$\begin{array}{ll} & f \text{ is monotonic} \\ \iff & \forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcap C') \sqsubseteq \bigsqcap \{f(c) \,|\, c \in C'\} \\ \iff & \forall\, \emptyset \neq C' \subseteq \mathcal{C}.\ f(\bigsqcup C') \sqsupseteq \bigsqcup \{f(c) \,|\, c \in C'\} \end{array}$$

Relating Monotonicity, Distributivity, Additivity

Let $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \perp, \top)$ be a complete lattice, $f : \mathcal{C} \to \mathcal{C}$ a function on $\mathcal{C}$.

Lemma 7.1.2.7

1. f is distributive iff f is additive.
2. f is monotonic if f is distributive or additive.

Chapter 7.2

Local DFA Semantics

Local DFA Semantics

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be an edge-labelled flow graph.

Definition 7.2.1 (Local DFA Semantics)

A local abstract DFA semanctics for G is a map

$$[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$$

where $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \perp, \top)$ is a complete lattice.

Note: The elements of $\widehat{\mathcal{C}}$ are the mathematical objects modeling and representing the data flow information of interest.

Chapter 7.3

DFA Specification

DFA Specification

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be an edge-labelled flow graph.

Definition 7.3.1 (DFA Specification)

A DFA specification for G is a triple $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_\mathbf{s})$ with

- $\widehat{\mathcal{C}} = (\mathcal{C}, \sqsubseteq, \sqcap, \sqcup, \perp, \top)$ a complete lattice.
- $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ a local abstract semantics.
- $c_\mathbf{s} \in \mathcal{C}$ an initial information (or start assertion).

Definition 7.3.2 (DFA Problem)

A DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_\mathbf{s})$ defines a DFA problem for G.

Note

Let $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification for G. Then:

- The elements of $\mathcal{C}$ represent the data flow information of interest.
- The functions $[\![\, e \,]\!]$, $e \in E$, abstract the concrete semantics of instructions to the level of the analysis.
- $c_{\mathbf{s}} \in \mathcal{C}$ is the data flow information assumed to be valid at the startnode **s** of G.

Overall, this gives rise to call

- $\widehat{C}$ a DFA lattice.
- $[\![\ ]\!]$ a local abstract DFA semantics (or DFA semantics).
- $[\![\, e \,]\!]$, $e \in E$, a local semantic DFA function (or DFA function).
- $c_{\mathbf{s}} \in \mathcal{C}$ a DFA start assertion.

General Convention

...for DFA lattices:

- greater in the lattice means better, more precise information!

(Note: In the Theory of Abstract Interpretation, this convention is made oppositely (cf. Chapter 15.2)

Example: Availability of a Term t (1)

...a term t is available at a program point n, if t is computed along every path p from **s** to n without that any operand of t is modified after the last computation of t on p.

DFA Specification for the Availability of a Term t:

- DFA lattice
  $$\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr}) = \widehat{\mathbb{B}}$$
- DFA semantics
  $[\![\ ]\!]_{av}^{t} : E \to (\mathbb{B} \to \mathbb{B})$ where
  $$\forall\, e \in E.\ \forall\, b \in \mathbb{B}.\ [\![e]\!]_{av}^{t}(b) =_{df} (b \vee Comp_e^t) \wedge Transp_e^t$$
- DFA start assertion: $b_{\mathbf{s}} \in \mathbb{B}$

Overall:

- Availability Specification: $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^{t}, b_{\mathbf{s}})$

Example: Availability of a Term t (2)

...where $\widehat{\mathbb{B}}$ denotes the data flow lattice and $Comp_e^t$, Mod_e^t, and $Transp_e^t$ three local predicates associated with edges and their instructions:

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr})$
  ...lattice of Boolean truth values: least element **falsch**, greatest element **wahr**, **falsch** $\leq$ **wahr**, logical $\wedge$ and logical $\vee$ as meet and join operation, respectively.
- $Comp_e^t$ ...**wahr**, if t is computed by the instruction at edge e, otherwise **falsch**.
- $Transp_e^t$ ...**wahr**, if e is transparent for t (i.e., no operand of t is assigned a new value by the instruction at edge e), otherwise **falsch**.

DFA Problems

...are practically relevant, if their underlying local DFA semantics are

- monotonic
- distributive/additive

and their data flow lattices satisfy the

- descending/ascending chain condition.

Properties of DFA Semantics, DFA Problems

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification for G.

Definition 7.3.3 (Properties of DFA Semantics)

The local DFA semantics $[\![\ ]\!] : E \to (\mathcal{C} \to \mathcal{C})$ of $\mathcal{S}_G$ is monotonic/distributive/additive iff all DFA functions $[\![\, e \,]\!]$, $e \in E$, are monotonic/distributive/additive, respectively.

Definition 7.3.4 (Properties of DFA Problems)

The DFA problem specified by $\mathcal{S}_G$

- is monotonic/distributive/additive iff the local DFA semantics $[\![\ ]\!]$ of $\mathcal{S}_G$ is monotonic/distributive/additive, respectively.
- satisfies the descending/ascending chain condition iff the DFA lattice $\widehat{C}$ of $\mathcal{S}_G$ satisfies the descending/ascending chain condition, respectively.

Example: Availability of a Term t (1)

Lemma 7.3.5 (DFA Functions)

$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av} \;=\; \begin{cases} Cst_{\mathbf{wahr}} & \text{if } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{B}} & \text{if } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{otherwise} \end{cases}$$

where

- $Cst_{\mathbf{wahr}},\ Cst_{\mathbf{falsch}} : \mathbb{B} \to \mathbb{B}$ (constant functions on $\mathbb{B}$)

  $Cst_{\mathbf{wahr}} =_{df} \lambda b.\ \mathbf{wahr}$

  $Cst_{\mathbf{falsch}} =_{df} \lambda b.\ \mathbf{falsch}$

- $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$ (identity on $\mathbb{B}$)

  $Id_{\mathbb{B}} =_{df} \lambda b.\ b$

Example: Availability of a Term t (2)

Lemma 7.3.6 (Chain Condition)

$\widehat{\mathbb{B}}$ satisfies the descending and ascending chain condition.

Lemma 7.3.7 (Distributivity, Additivity)

$[\![\, e \,]\!]_{av}^{t}$, $e \in E$, is distributive and additive (and hence, also monotonic).

Proof. Immediately with Lemma 7.3.5 and Lemma 7.1.2.7(2).

Corollary 7.3.8 (Availability of a Term t)

The DFA problem specified by $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\;]\!]_{av}^{t}, b_{\mathbf{s}})$ is distributive and additive and satisfies the descending and ascending chain condition.

Towards a Global Abstract Semantics

...by globalizing a local abstract semantics for instructions to a global abstract semantics for flow graphs.

This leads to the nondeterministic operational

- collecting (CS) semantics

from which we derive two deterministic operational variants:

- The meet over all paths (MOP) semantics
- The join over all paths (JOP) semantics

...together with two computational deterministic denotational variants:

- The maximum fixed point ($MaxFP$) semantics
- The minimum fixed point ($MinFP$) semantics

which induce computation procedures for computing or approximating their operational counterparts.

Chapter 7.4

Operational Global DFA Semantics

Chapter 7.4.1

Collecting Semantics

Extending DFA Functions from Edges to Paths

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

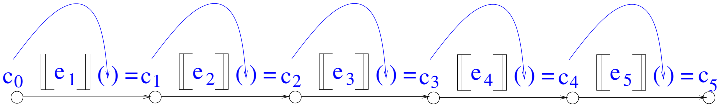
Definition 7.4.1.1 (Extending $[\![\ ]\!]$ to Paths)

The DFA semantics $[\![\, e \,]\!]$, $e \in E$, is extended from edges onto paths $p = \langle e_1, e_2, \ldots, e_q \rangle$ by defining:

$$[\![\, p \,]\!] =_{df} \begin{cases} Id_{\mathcal{C}} & \text{if } \lambda_p < 1 \\ [\![\, \langle e_2, \ldots, e_q \rangle \,]\!] \circ [\![\, e_1 \,]\!] & \text{otherwise} \end{cases}$$

where $Id_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$ denotes the identity on $\mathcal{C}$, i.e., $Id_{\mathcal{C}} = \lambda c.\ c$.

Illustrating the extension of $[\![\ ]\!]$ from edges to paths:



The Collecting DFA Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_\mathbf{s})$ be a DFA specification.

Definition 7.4.1.2 (Collecting DFA Semantics)

The (nondeterministic) collecting DFA semantics (or global abstract semantics) induced by $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{CS}_{\mathcal{S}_G} : N \rightarrow \mathcal{P}(\mathcal{C})$$

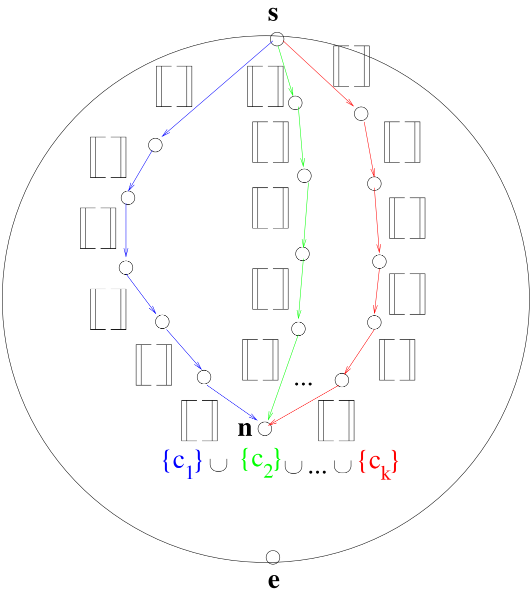
$$\forall\, n \in N.\ [\![\, n\,]\!]^{CS}_{\mathcal{S}_G} =_{df} \{\, [\![\, p\,]\!](c_\mathbf{s}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\}$$

where $\mathcal{P}$ denotes the powerset operator.

Note:

$$[\![\, \mathbf{s}\,]\!]^{CS}_{\mathcal{S}_G} \;=\; \{c_\mathbf{s}\}$$

Illustrating the Collecting DFA Semantics



Note

...if π is a $\textsc{While}$ program, G its flow graph representation, and $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], \bot)$ a DFA specification for G wrt the start assertion $\bot$, then the global DFA semantics at the program end node $\mathbf{e}$

$$[\![\, \mathbf{e} \,]\!]^{CS}_{\mathcal{S}_G} \; = \; \{\, [\![\, p \,]\!](\bot) \mid p \in \mathbf{P}[\mathbf{s}, \mathbf{e}] \,\}$$

can be considered the nondeterministic abstract counterpart of the deterministic $\textsc{While}$ semantics of π for Σ:

$$[\![\, \pi \,]\!]_{\textsc{While}}(\Sigma) \; = \{\, [\![\, \pi \,]\!]_{\textsc{While}}(\sigma) \mid \sigma \in \Sigma \}$$

Chapter 7.4.2

The Meet Over All Paths Semantics

The Meet Over All Paths (*MOP*) Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Definition 7.4.2.1 (*MOP* Semantics)

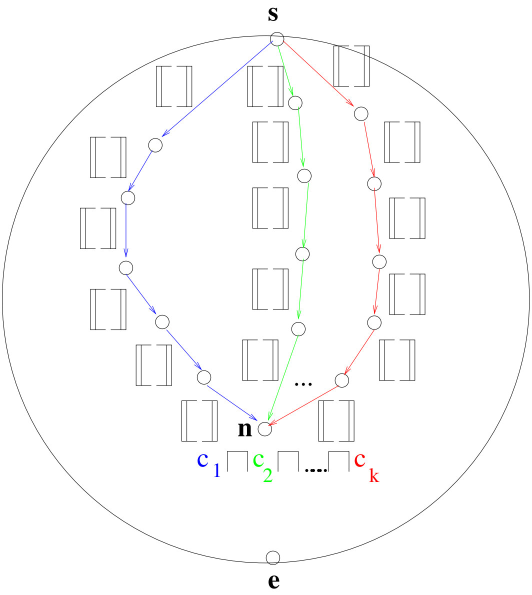
The (deterministic) *MOP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{MOP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N.\ [\![\, n\,]\!]^{MOP}_{\mathcal{S}_G} &=_{df} \bigsqcap [\![\, n\,]\!]^{CS}_{\mathcal{S}_G} \\ &= \bigsqcap \{\, [\![\, p\,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n]\, \} \end{aligned}$$

Note: $\bigsqcap [\![\, n\,]\!]^{CS}_{\mathcal{S}_G}$ and hence $[\![\, n\,]\!]^{MOP}_{\mathcal{S}_G}$, $n \in N$, exists, since $\widehat{\mathcal{C}}$ is a complete lattice.

Illustrating the *MOP* Semantics



Chapter 7.4.3

The Join Over All Paths Semantics

The Join Over All Paths (*JOP*) Semantics

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Definition 7.4.3.1 (*JOP* Semantics)

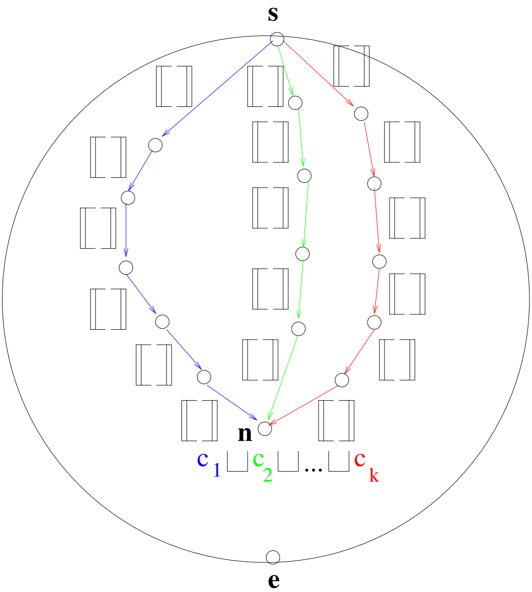
The (deterministic) *JOP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]^{JOP}_{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\begin{aligned} \forall\, n \in N.\ [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G} \quad &=_{df} \quad \bigsqcup [\![\, n \,]\!]^{CS}_{\mathcal{S}_G} \\ &= \quad \bigsqcup \{\, [\![\, p \,]\!](c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \,\} \end{aligned}$$

Note: $\bigsqcup [\![\, n \,]\!]^{CS}_{\mathcal{S}_G}$ and hence $[\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$, $n \in N$, exists, since $\widehat{\mathcal{C}}$ is a complete lattice.

Illustrating the *JOP* Semantics



Chapter 7.4.4

MOP and *JOP* Semantics as Specifying Solutions of DFA Problems

As illustrated by the Figures

...of Chapter 7.4.2 and 7.4.3, the *MOP* and the *JOP* semantics bound for program point n the DFA information

- possible at n wrt $\mathcal{S}_G$:

Independently of the path $p \in \mathbf{P}[\mathbf{s}, n]$ along which node n is reached, the information provided by p at n is

- at least as large as the *MOP* semantics at n (it can not be worse, only better):

$$[\![\, p \,]\!](c_{\mathbf{s}}) \sqsupseteq [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G}$$

- at most as large as the *JOP* semantics at n (it can not be better, only worse):

$$[\![\, p \,]\!](c_{\mathbf{s}}) \sqsubseteq [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$$

This means:

$$\forall \, p \in \mathbf{P}[\mathbf{s}, n]. \; [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G} \sqsubseteq [\![\, p \,]\!](c_{\mathbf{s}}) \sqsubseteq [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$$

In other words

...the *MOP* and the *JOP* semantics provide for every program point n the DFA informations which are

- the best possible valid ones at n wrt $\mathcal{S}_G$

in the following sense:

- $[\![\, n \,]\!]_{\mathcal{S}_G}^{MOP}$ is the minimum information valid at n (it can not be worse, only better): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![\, n \,]\!]_{\mathcal{S}_G}^{MOP} \sqsubseteq [\![\, p \,]\!](c_{\mathbf{s}})$.
- $[\![\, n \,]\!]_{\mathcal{S}_G}^{JOP}$ is the maximum information valid at n (it can not be better, only worse): $\forall\, p \in \mathbf{P}[\mathbf{s}, n].\ [\![\, n \,]\!]_{\mathcal{S}_G}^{JOP} \sqsupseteq [\![\, p \,]\!](c_{\mathbf{s}})$.

This means, the *MOP* and *JOP* semantics ensure the absence of 'surprises:' Independently of the path $p \in \mathbf{P}[\mathbf{s}, n]$ taken along which node n is reached, we always have:

$$[\![\, n \,]\!]_{\mathcal{S}_G}^{MOP} \sqsubseteq [\![\, p \,]\!](c_{\mathbf{s}}) \sqsubseteq [\![\, n \,]\!]_{\mathcal{S}_G}^{JOP}$$

The Specifying Solutions of a DFA Problem

This gives rise to the following definition:

Definition 7.4.4.1 (Specifying Solutions of a DFA P.)

The *MOP* and the *JOP* semantics of a flow graph define the specifying solutions of a DFA problem, its so-called *MOP* and *JOP* solutions.

Conservative DFA Algorithms

Definition 7.4.4.2 (Conservative DFA Algorithm)

A DFA algorithm A is

- *MOP* conservative
- *JOP* conservative

for $\mathcal{S}_G$, if A terminates with

- a lower approximation of the *MOP* semantics
- an upper approximation of the *JOP* semantics

of $\mathcal{S}_G$, respectively.

Tight DFA Algorithms

Definition 7.4.4.3 (Tight DFA Algorithm)

A DFA algorithm A is

- *MOP* tight
- *JOP* tight

for $\mathcal{S}_G$, if A terminates with

- the *MOP* semantics
- the *JOP* semantics

of $\mathcal{S}_G$, respectively.

Chapter 7.4.5

Undecidability of *MOP* and *JOP* Semantics

Unfortunately

...the definitions of the *MOP* and *JOP* semantics do not directly induce

- effective computation procedures

for computing them (think of loops in a non-deterministically interpreted flow graph causing the number of paths reaching a node to be infinite).

Even worse, the *MOP* and *JOP* semantics of a flow graph are

- not decidable!

Undecidability of the *MOP* Semantics

Theorem 7.4.5.1 (Undecidability of the *MOP* Sem.)

There is no algorithm A such that:

- The input of A is
  - a DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$.
  - algorithms for computing of the meet, the equality test, and the application of monotonic functions on $\widehat{\mathcal{C}}$.
- The output of A is the *MOP* semantics of $\mathcal{S}_G$.

(John B. Kam, Jeffrey D. Ullman. Monotone Data Flow Analysis Frameworks. Acta Informatica 7:305-317, 1977)

Corollary 7.4.5.2 (Undecidability of the *JOP* Sem.)

There is no algorithm A such that:

- The input of A is
  - a DFA specification $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.
  - algorithms for the computing the meet, the equality test, and the application of monotonic functions on $\widehat{\mathcal{C}}$.
- The output of A is the *JOP* semantics of $\mathcal{S}_G$.

Towards Conservative and Tight DFA Alg's

...because of the preceding negative results we complement the operational approach underlying the *MOP* and *JOP* semantics an orthogonal denotational globalization approach of a local abstract semantics leading to the

- Maximum fixed point (*MaxFP*) semantics
- Minimum fixed point (*MinFP*) semantics

of a flow graph, respectively.

The *MaxFP* and *MinFP* semantics are also called the

- Maximum fixed point (*MaxFP*) solution
- Minimum fixed point (*MinFP*) solution

of a DFA problem, respectively, which (under certain conditions) can

- effectively be computed.

Chapter 7.5

Denotational Global DFA Semantics

Chapter 7.5.1

The Maximum Fixed Point Semantics

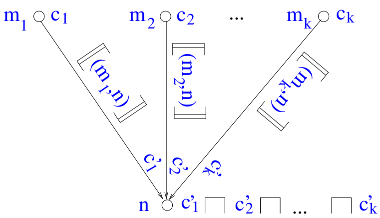
The Maximum Fixed Point (*MaxFP*) Approach

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Equation System 7.5.1.1 (*MaxFP* Equation System)

$$inf(n) \;=\; \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcap \{\, [\![\, (m,n) \,]\!](inf(m)) \mid m \in pred(n) \,\} & \text{otherwise} \end{cases}$$

Illustrating the *MaxFP* Approach ($n \neq \mathbf{s}$):



The *MaxFP* Semantics

Let

- $\nu\text{-}inf_{c_s}(n),\ n \in N$

denote the greatest solution of Equation System 7.5.1.1.

Definition 7.5.1.2 (*MaxFP* Semantics)

The (deterministic) *MaxFP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]_{\mathcal{S}_G}^{MaxFP} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ [\![\, n \,]\!]_{\mathcal{S}_G}^{MaxFP} =_{df} \nu\text{-}inf_{c_s}(n)$$

Note:

$$[\![\, \mathbf{s} \,]\!]_{\mathcal{S}_G}^{MaxFP} \;=\; c_{\mathbf{s}}$$

Chapter 7.5.2

The Minimum Fixed Point Semantics

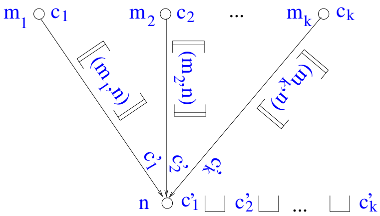
The Minimum Fixed Point (*MinFP*) Approach

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Equation System 7.5.2.1 (*MinFP* Equation System)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcup \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Illustrating the *MinFP* Approach ($n \neq \mathbf{s}$):



The *MinFP* Semantics

Let

- $\mu\text{-}inf_{c_{\mathbf{s}}}(n),\ n \in N$

denote the least solution of Equation System 7.5.2.1.

Definition 7.5.2.2 (*MinFP* Semantics)

The *MinFP* semantics of $\mathcal{S}_G$ is defined by:

$$[\![\ ]\!]_{\mathcal{S}_G}^{MinFP} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ [\![\, n \,]\!]_{\mathcal{S}_G}^{MinFP} =_{df} \mu\text{-}inf_{c_{\mathbf{s}}}(n)$$

Note:

$$[\![\, \mathbf{s} \,]\!]_{\mathcal{S}_G}^{MinFP} \;=\; c_{\mathbf{s}}$$

Chapter 7.6

The Generic Fixed Point Algorithm

The *MaxFP* and *MinFP* Semantics

...are practically relevant because *MaxFP* Equation System 7.5.1.1 and *MinFP* Equation System 7.5.2.1 induce a generic

- iterative computation procedure (Algorithm 7.6.1.1)

approximating their greatest and least solutions, respectively, i.e., the *MaxFP* and *MinFP* semantics.

Chapter 7.6.1

Algorithm

The Generic Fixed Point Algorithm 7.6.1.1 (1)

Input: A DFA specification $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$.

Output: On termination of the algorithm (cf. Termination Theorem 7.6.2.1), variable $inf[n]$ stores the *MaxFP* solution of $\mathcal{S}_G$ at node n.

Additionally (cf. Safety Theorem 7.7.1 and Coincidence Theorem 7.7.2): If $[\![\ ]\!]$ is

- distributive: $inf[n]$ stores
- monotonic: $inf[n]$ stores a lower approximation of

the *MOP* solution of $\mathcal{S}_G$ at node n.

Remark: The variable *workset* controls the iterative process. It temporarily stores a set of nodes of G, whose annotations have recently been changed and thus can impact the annotations of their neighbouring nodes.

The Generic Fixed Point Algorithm 7.6.1.1 (2)

(Prologue: Initializing *inf* and *workset*)

FORALL $n \in N \backslash \{\mathbf{s}\}$ DO $inf[n] := \top$ OD;
$inf[\mathbf{s}] := c_{\mathbf{s}}$;
$workset := N$;

(Main loop: The iterative fixed point computation)

WHILE $workset \neq \emptyset$ DO
    CHOOSE $m \in workset$;
        $workset := workset \backslash \{ m \}$;
        (Updating the annotations of all successors of node m)
        FORALL $n \in succ(m)$ DO
            $meet := [\![(m,n)]\!](inf[m]) \sqcap inf[n]$;
            IF $inf[n] \sqsupset meet$
                THEN
                    $inf[n] := meet$;
                    $workset := workset \cup \{ n \}$
            FI
        OD ESOOHC OD.

Chapter 7.6.2

Termination

Termination

Theorem 7.6.2.1 (Termination)

The Generic Fixed Point Algorithm 7.6.1.1 terminates w/ the

1. *MaxFP* semantics of $\mathcal{S}_G$, if
   1.1 $[\![\ ]\!]$ is monotonic
   1.2 $\widehat{\mathcal{C}}$ satisfies the descending chain condition.
2. *MinFP* semantics of $\mathcal{S}_G$, if
   2.1 $[\![\ ]\!]$ is monotonic
   2.2 $\widehat{\mathcal{C}^{usd}}$ satisfies the ascending chain condition, where

$$\widehat{\mathcal{C}^{usd}} =_{df} (\mathcal{C}, \sqcup, \sqcap, \sqsupseteq, \top, \bot)$$

is lattice $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top)$ put up-side down.

The Computable Solutions of a DFA Problem

...together the Generic Fixed Point Algorithm 7.6.1.1 and Termination Theorem 7.6.2.1 give rise to the following definition:

Definition 7.6.2.2 (Computable Solutions of a DFA P.)

The *MaxFP* and the *MinFP* semantics of a flow graph define the computable solutions of a DFA problem, its so-called *MaxFP* and *MinFP* solutions.

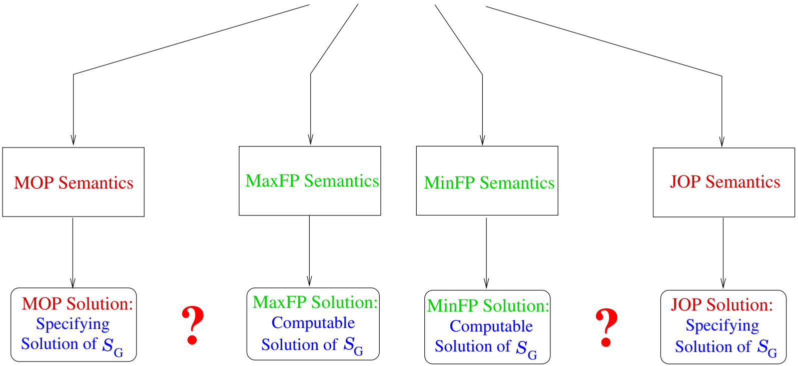
Chapter 7.7

Safety and Coincidence

MOP / *MaxFP*- and *JOP* / *MinFP* Semantics

...of a DFA specification and the question of their relationship:

$$S_G =_{df} (\widehat{C}, [\![\ ]\!], c_s)$$



Safety

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ be a DFA specification.

Theorem 7.7.1 (Safety)

1. The *MaxFP* semantics of $\mathcal{S}_G$ is a safe (i.e., lower) approximation of the *MOP* semantics of $\mathcal{S}_G$, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MaxFP}_{\mathcal{S}_G} \sqsubseteq [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G}$$

2. The *MinFP* semantics of $\mathcal{S}_G$ is a safe (i.e., upper) approximation of the *JOP* semantics of $\mathcal{S}_G$, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]^{MinFP}_{\mathcal{S}_G} \sqsupseteq [\![\, n \,]\!]^{JOP}_{\mathcal{S}_G}$$

if the DFA semantics $[\![\ ]\!]$ is monotonic.

Coincidence

Let $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!], c_{\mathbf{s}})$ be a DFA specification.

Theorem 7.7.2 (Coincidence)

1. The *MaxFP* and the *MOP* semantics of $\mathcal{S}_G$ coincide, i.e.,

$$\forall\, n \in N.\ [\![\, n \,]\!]_{\mathcal{S}_G}^{MaxFP} = [\![\, n \,]\!]_{\mathcal{S}_G}^{MOP}$$

2. The *MinFP* and the *JOP* semantics of $\mathcal{S}_G$ coincide, i.e.,

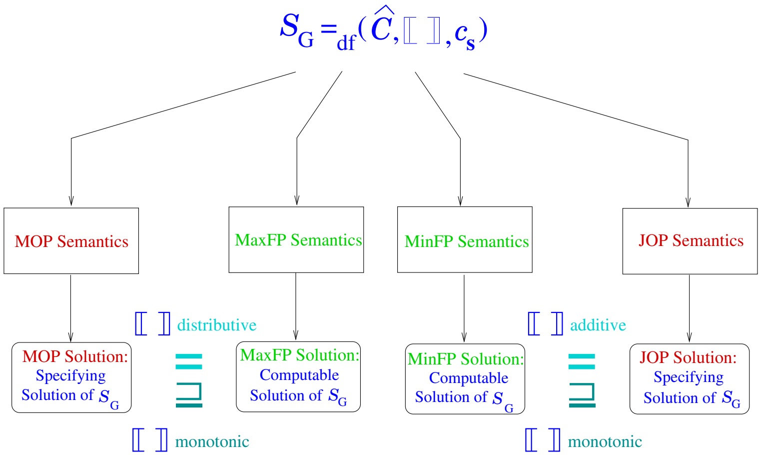
$$\forall\, n \in N.\ [\![\, n \,]\!]_{\mathcal{S}_G}^{MinFP} = [\![\, n \,]\!]_{\mathcal{S}_G}^{JOP}$$

if the DFA semantics $[\![\ ]\!]$ is distributive or additive, respectively.

Recall Lemma 7.1.2.7(1): $[\![\ ]\!]$ is distributive iff $[\![\ ]\!]$ is additive.

MOP / *MaxFP*- and *JOP* / *MinFP* Semantics

...of a DFA Specification and their relationship:



Conservativity of Algorithm 7.6.1.1

Corollary 7.7.3 (*MOP*/*JOP* Conservativity)

Algorithm 7.6.1.1 is

- *MOP* (*JOP*) conservative

for $\mathcal{S}_G$, i.e., it terminates with a lower (upper) approximation of the *MOP* (*JOP*) semantics of $\mathcal{S}_G$, if

1. $[\![\ ]\!]$ is monotonic
2. $\widehat{\mathcal{C}}$ satisfies the descending (ascending) chain condition, respectively.

Tightness of Algorithm 7.6.1.1

Corollary 7.7.4 (*MOP*/*JOP* Tightness)

Algorithm 7.6.1.1 is

- *MOP* (*JOP*) tight

for $\mathcal{S}_G$, i.e., it terminates with the *MOP* (*JOP*) semantics of $\mathcal{S}_G$, if

1. $[\![\ ]\!]$ is distributive (additive)
2. $\widehat{\mathcal{C}}$ satisfies the descending (ascending) chain condition

respectively.

Chapter 7.8

Soundness and Completeness

Soundness and Completeness (1)

Analysis Scenario:

- Let ϕ be a program property of interest (e.g., availability of a term, liveness of a variable, etc.).
- Let $\mathcal{S}_G^\phi$ be a DFA specification designed for ϕ.

Definition 7.8.1 (Soundness)

$\mathcal{S}_G^\phi$ is *MOP* sound (*JOP* sound) for ϕ, if, whenever the *MOP* semantics (*JOP* semantics) of $\mathcal{S}_G^\phi$ indicates that ϕ is valid, then ϕ is valid.

Definition 7.8.2 (Completeness)

$\mathcal{S}_G^\phi$ is *MOP* complete (*JOP* complete) for ϕ, if, whenever ϕ is valid, then the *MOP* semantics (*JOP* semantics) of $\mathcal{S}_G^\phi$ indicates that ϕ is valid.

Soundness and Completeness (2)

Intuitively

- *MOP* soundness means: $[\![\ ]\!]^{MOP}_{\mathcal{S}^{\phi}_{G}}$ 'implies' ϕ.
- *MOP* completeness means: ϕ 'implies' $[\![\ ]\!]^{MOP}_{\mathcal{S}^{\phi}_{G}}$.

and

- *JOP* soundness means: ϕ 'implies' $[\![\ ]\!]^{JOP}_{\mathcal{S}^{\phi}_{G}}$.
- *JOP* completeness means: $[\![\ ]\!]^{JOP}_{\mathcal{S}^{\phi}_{G}}$ 'implies' ϕ.

Soundness and Completeness (3)

Intuitively, if $\mathcal{S}_G^\phi$ is *MOP* (*JOP*) sound and complete for ϕ, this means:

We compute

- the property of interest,
- the whole property of interest,
- and only the property of interest.

In other words, we compute

- the program property of interest accurately!

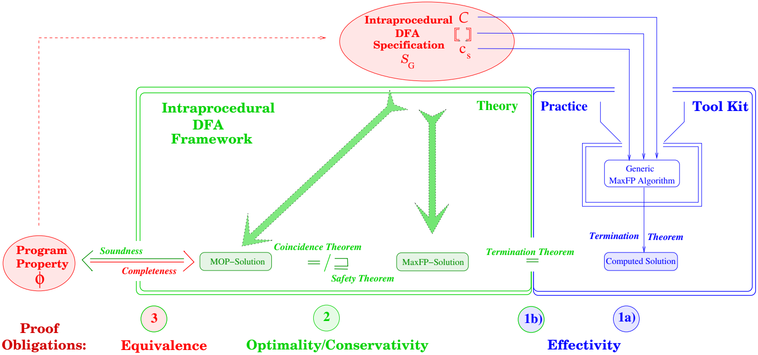
Chapter 7.9

The Framework and Toolkit View of DFA

Data Flow Analysis: A Holistic View

...considering (intraprocedural) DFA from a holistic angle: The

- Framework and Toolkit (*MOP*/*MaxFP*) View of DFA



Data Flow Analysis in Practice

...working with framework and toolkit is a three-stage process:

The Three-Stage Process

1. Identifying a Program Property of Interest

   Identify a program property of interest (e.g., availability of a term, liveness of a variable, etc.), say ϕ, and define ϕ formally.

2. Designing a DFA Specification

   Design a DFA specification $\mathcal{S}_G^{\phi} = (\widehat{\mathcal{C}}, [\![\ ]\!], c_s)$ for ϕ.

3. Accomplishing Proof Obligations, Obtaining Guarantees

   Prove a fixed set of proof obligations about the components of $\mathcal{S}_G^{\phi}$ and the relation of its *MOP* (*JOP*) solution and ϕ to obtain guarantees that its *MaxFP* (*MinFP*) solution is sound or even sound and complete for ϕ.

Proof Obligations, Implied Guarantees (1)

Proof obligations and guarantees in detail:

- Proof Obligations 1a), 1b): Descending (ascending) chain condition for $\widehat{\mathcal{C}}$, monotonicity for $[\![\ ]\!]$

  Guarantees:

  - Effectivity: Termination of Algorithm 7.6.1.1 with the *MaxFP* (*MinFP*) semantics of $\mathcal{S}_G^{\phi}$.
  - Conservativity: The *MaxFP* (*MinFP*) solution of $\mathcal{S}_G^{\phi}$ is *MOP* (*JOP*) conservative.

- Proof Obligation 2): Distributivity (additivity) for $[\![\ ]\!]$

  Guarantee:

  - Tightness: The *MaxFP* (*MinFP*) semantics of $\mathcal{S}_G^{\phi}$ is *MOP* (*JOP*) tight.

Proof Obligations, Implied Guarantees (2)

- Proof Obligation 3): Equivalence of $MOP_{\mathcal{S}_G^\phi}$ ($JOP_{\mathcal{S}_G^\phi}$) and ϕ

  Guarantees:

  - Whenever the *MOP* solution of $\mathcal{S}_G^\phi$ indicates the validity of ϕ, then it is valid: Soundness.

    $\leadsto$ We compute the property of interest, and only the property of interest.
  - Whenever ϕ is valid, this is indicated by the *MOP* solution of $\mathcal{S}_G^\phi$: Completeness.

    $\leadsto$ We compute the whole property of interest.
  - Vice versa for the *JOP* solution of $\mathcal{S}_G^\phi$.

  Guarantee of combined Soundness and Completeness:

  - We compute program property ϕ accurately!

Chapter 7.10

Applications

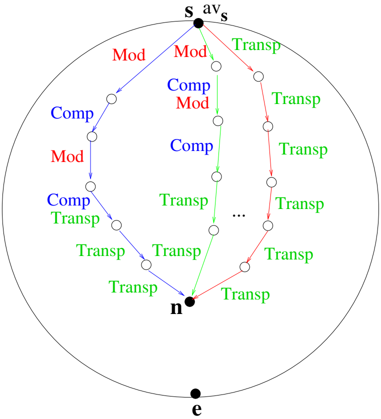
Chapter 7.10.1

Distributive DFA: Available Expressions

Intuitively

...a term is available at a node if, no matter which path is taken from the entry of the program to that node, the term is computed without that any of the variables occurring in it is redefined before reaching this node.

Illustration:



Availability

...we will specify the availability problem in four different variants in order to illustrate the

- usage of different lattices for DFA
- class of so-called
  - bitvector
  - Gen/Kill

  DFA problems availability is a typical representative of.

...computing available terms is a

- canonical example of a distributive DFA problem.

Preliminaries

Let $\iota_e \equiv x := exp$ (or $\iota_e \equiv exp$) be the instruction (or the condition) at edge e, t a term.

Local Predicates (for edges)

- $Comp_e^t$

  ...**wahr**, if t is computed by ι_e (i.e., t is a subterm of the right-hand side expression exp of ι_e), otherwise **falsch**.

- Mod_e^t

  ...**wahr**, if t is modified by ι_e (i.e., ι_e assigns a new value to some operand of t), otherwise **falsch**.

- $Transp_e^t =_{df} \neg Mod_e^t$

  ...**wahr**, if e is transparent for t (i.e., ι_e does not assign a new value to any operand of t), otherwise **falsch**.

Variant 1: Fixing the Setting

...availability for a single term t.

Lattice

- $\widehat{\mathbb{B}} =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr})$

  ...lattice of Boolean truth values: least element **falsch**, greatest element **wahr**, **falsch** $\leq$ **wahr**, logical $\wedge$ and logical $\vee$ as meet and join operation, respectively.

Utility Functions

- Constant Functions $Cst_{\textbf{wahr}}, Cst_{\textbf{falsch}} : \mathbb{B} \to \mathbb{B}$

  $Cst_{\textbf{wahr}} =_{df} \lambda b.\ \textbf{wahr}$
  $Cst_{\textbf{falsch}} =_{df} \lambda b.\ \textbf{falsch}$

- Identity $Id_{\mathbb{B}} : \mathbb{B} \to \mathbb{B}$

  $Id_{\mathbb{B}} =_{df} \lambda b.\ b$

Variant 1: Specifying the DFA

DFA Specification

- DFA lattice
  $$\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathbb{B}, \wedge, \vee, \leq, \textbf{falsch}, \textbf{wahr}) = \widehat{\mathbb{B}}$$
- DFA semantics
  $[\![\ ]\!]^t_{av} : E \to (\mathbb{B} \to \mathbb{B})$ where
  $\forall\, e \in E\ \forall\, b \in \mathbb{B}.\ [\![\, e\,]\!]^t_{av}(b) =_{df} (b \vee Comp^t_e) \wedge Transp^t_e$
- Start assertion: $b_{\mathbf{s}} \in \mathbb{B}$

Availability Specification for t

- Specification: $\mathcal{S}^{av,t}_G = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, b_{\mathbf{s}})$

Variant 1: Fulfilling the Proof Obligations

Lemma 7.10.1.1 (DFA Functions)

$$\forall\, e \in E.\ [\![\, e \,]\!]^t_{av} = \begin{cases} Cst_{\mathbf{wahr}} & \text{if } Comp^t_e \wedge Transp^t_e \\ Id_{\mathbb{B}} & \text{if } \neg Comp^t_e \wedge Transp^t_e \\ Cst_{\mathbf{falsch}} & \text{otherwise} \end{cases}$$

Lemma 7.10.1.2 (Chain Conditions)

$\widehat{\mathbb{B}}$ satisfies the descending and ascending chain condition.

Lemma 7.10.1.3 (Distributivity, Additivity)

$[\![\ ]\!]^t_{av}$ is distributive and additive.

Proof. Immediately with Lemma 7.10.1.1.

Corollary 7.10.1.4 (Monotonicity)

$[\![\ ]\!]^t_{av}$ is monotonic.

Variant 1: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.5 (Termination)

Applied to $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,t}$.

Proof. Immediately with Lemma 7.10.1.2, Corollary 7.10.1.4, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.6 (Tightness)

Applied to $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]_{av}^t, b_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,t}$ (i.e., terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,t}$).

Proof. Immediately with Lemma 7.10.1.3, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Variant 2: Fixing the Setting

...availability for a finite set of terms T.

Lattice

- $\widehat{\mathcal{P}(T)} =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T)$

  ...power set lattice of T: least element $\emptyset$, greatest element T, subset relation $\subseteq$ as ordering relation, set intersection $\cap$ and set union $\cup$ as meet and join operation, respectively.

Variant 2: Specifying the DFA

DFA Specification

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA semantics
  $[\![\ ]\!]_{av}^{T} : E \to (\mathcal{P}(T) \to \mathcal{P}(T))$ where
  $\forall\, e \in E\ \forall\, T' \in \mathcal{P}(T).\ [\![\, e\,]\!]_{av}^{T}(T') =_{df}$
  $\{t \in T \mid (t \in T' \vee Comp_e^t) \wedge Transp_e^t\}$
- Start assertion: $T_{\mathbf{s}} \in \mathcal{P}(T)$

Availability Specification for T

- Specification: $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^{T}, T_{\mathbf{s}})$

Variant 2: Fulfilling the Proof Obligations

Lemma 7.10.1.7 (Chain Conditions)

$\widehat{\mathcal{P}(T)}$ satisfies the descending and ascending chain condition.

Lemma 7.10.1.8 (Distributivity, Additivity)

$[\![\ ]\!]_{av}^{T}$ is distributive and additive.

Corollary 7.10.1.9 (Monotonicity)

$[\![\ ]\!]_{av}^{T}$ is monotonic.

Variant 2: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.10 (Termination)

Applied to $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_s)$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,T}$.

Proof. Immediately with Lemma 7.10.1.7, Corollary 7.10.1.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.11 (Tightness)

Applied to $\mathcal{S}_G^{av,T} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av}^T, T_s)$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,T}$ (i.e., it terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,T}$).

Proof. Immediately with Lemma 7.10.1.8, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Variant 3: Fixing the Setting (1)

...availability for a finite set of terms T, $|T| = n$.

Lattice

- $\widehat{\mathbb{B}^n} =_{df} (\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\textbf{falsch}}, \overline{\textbf{wahr}})$

  ...*n*-ary cross-product lattice over $\mathbb{B}$: least element $\overline{\textbf{falsch}} =_{df} (\textbf{falsch}, \ldots, \textbf{falsch}) \in \mathbb{B}^n$, greatest element $\overline{\textbf{wahr}} =_{df} (\textbf{wahr}, \ldots, \textbf{wahr}) \in \mathbb{B}^n$, ordering relation $<_{pw}$ as pointwise extension of $<$ from $\widehat{\mathbb{B}}$ to $\widehat{\mathbb{B}^n}$, $\wedge_{pw}$ and $\vee_{pw}$ as pointwise extensions of logical $\wedge$ and logical $\vee$ from $\widehat{\mathbb{B}}$ to $\widehat{\mathbb{B}^n}$ as meet and join operation, respectively.

Variant 3: Fixing the Setting (2)

Utility Functions

- $ix : T \to \{1, \ldots, n\}, \; ix^{-1} : \{1, \ldots, n\} \to T$

  ...bijective index mappings which uniquely associates every term $t \in T$ with a number in $\{1, \ldots, n\}$ and vice versa.

  The $ix(t)^{th}$ element of an element
  $\bar{b} = (b_1, \ldots, b_{ix(t)}, \ldots, b_n) \in \mathbb{B}^n$
  is the availability information for t stored in $\bar{b}$.

- $\cdot \downarrow_{\cdot} : \mathbb{B}^n \to \{1, \ldots, n\} \to \mathbb{B}$

  ...projection function which yields the i^{th} element of an element $\bar{b} \in \mathbb{B}^n$, i.e., $\forall\, i \in \{1, \ldots, n\}.\ \bar{b} \downarrow_i =_{df} b_i$.

Variant 3: Specifying the DFA (cross-pr. view)

DFA Specification (cross-product view (cpv))

- DFA lattice
  $$\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$$
  $$(\mathbb{B}^n, \wedge_{pw}, \vee_{pw}, <_{pw}, \overline{\textbf{falsch}}, \overline{\textbf{wahr}}) = \widehat{\mathbb{B}^n}$$
- DFA semantics
  $[\![\ ]\!]^T_{av,cpv} : E \rightarrow (\mathbb{B}^n \rightarrow \mathbb{B}^n)$ where
  $\forall\, e \in E\ \forall\, v \in \mathbb{B}^n.\ [\![\, e\,]\!]^T_{av,cpv}(\bar{b}) =_{df} \bar{b}'$
  where $\forall\, i \in \{1, \ldots, n\}.\ \bar{b}'\!\downarrow_i\ =_{df}$
  $$(\bar{b}\!\downarrow_i\ \vee\ Comp_e^{ix^{-1}(i)}) \wedge Transp_e^{ix^{-1}(i)}$$
- Start assertion: $\bar{b}_{\mathbf{s}} \in \mathbb{B}^n$

Availability Specification for T

- Specification: $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]^T_{av,cpv}, \bar{b}_{\mathbf{s}})$

Variant 3: Towards the Bitvector View (1)

...as implementation of $\mathcal{S}_G^{av,T,cpv}$:

- $\widehat{\mathbb{B}^n}$ can efficiently be implemented in terms of bitvectors $\vec{bv} = [d_1, \ldots, d_n]$, $d_i \in \{0, 1\}$, $1 \leq i \leq n$, of length n.
- Let $\mathcal{BV}^n$ denote the set of all bitvectors of length n.
- Let $\vec{bv}[i] = d_i$ for all $\vec{bv} = [d_1, \ldots, d_n] \in \mathcal{BV}^n$, $1 \leq i \leq n$.
- Let $\vec{0} =_{df} [0, \ldots, 0] \in \mathcal{BV}^n$ and $\vec{1} =_{df} [1, \ldots, 1] \in \mathcal{BV}^n$.
- Let $min_{\mathcal{BV}}$ and $max_{\mathcal{BV}}$ be the bitwise minimum ('logical $\wedge$') and the bitwise maximum function ('logical $\vee$') on bitvectors, i.e., $\forall\, \vec{bv}_1, \vec{bv}_2 \in \mathcal{BV}^n\ \forall\, i \in \{1, \ldots, n\}$.
  - $(\vec{bv}_1\ min_{\mathcal{BV}}\ \vec{bv}_2)[i] =_{df} min(\vec{bv}_1[i], \vec{bv}_2[i])$
  - $(\vec{bv}_1\ max_{\mathcal{BV}}\ \vec{bv}_2)[i] =_{df} max(\vec{bv}_1[i], \vec{bv}_2[i])$

Variant 3: Towards the Bitvector View (2)

Utility Functions:

- $ix : T \to \{1, \ldots, n\},\ ix^{-1} : \{1, \ldots, n\} \to T$

  ...bijective index mappings which associate every term $t \in T$ uniquely with a number in $\{1, \ldots, n\}$ and vice versa.

  The $ix(t)^{th}$ element of a bitvector
  $\vec{bv} = [d_1, \ldots, d_{ix(t)}, \ldots, d_n)] \in \mathcal{BV}^n$
  is the availability information for t stored in $\vec{bv}$.

Variant 3: Towards the Bitvector View (3)

...extending and adapting local predicates to bitvectors:

- $\overrightarrow{Comp_e^T} \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Comp_e^T}\,[i] =_{df} \begin{cases} 1 & \text{if } Comp_e^{ix^{-1}(i)} \\ 0 & \text{otherwise} \end{cases}$$

- $\overrightarrow{Transp_e^T} \in \mathcal{BV}^n$

  $$\forall\, i \in \{1, \ldots, n\}.\ \overrightarrow{Transp_e^T}\,[i] =_{df} \begin{cases} 1 & \text{if } Transp_e^{ix^{-1}(i)} \\ 0 & \text{otherwise} \end{cases}$$

Variant 3: Specifying the DFA (bitvector view)

DFA Specification (bitvector view (bvv))

- DFA lattice
  $$\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top) =_{df}$$
  $$(\mathcal{BV}^n, min_{\mathcal{BV}}, max_{\mathcal{BV}}, <_{\mathcal{BV}}, \vec{0}, \vec{1}) = \widehat{\mathcal{BV}^n}$$
- DFA semantics
  $[\![\ ]\!]^T_{av,bvv} : E \to (\mathcal{BV}^n \to \mathcal{BV}^n)$ where
  $$\forall\, e \in E\ \forall\, \vec{bv} \in \mathcal{BV}^n.\ [\![\, e\,]\!]^T_{av,bvv}(\vec{bv}) =_{df}$$
  $$(\vec{bv}\ \ max_{\mathcal{BV}}\ \ \overrightarrow{Comp}^T_e)\ min_{\mathcal{BV}}\ \ \overrightarrow{Transp}^T_e$$
- Start assertion: $\vec{bv}_{\mathbf{s}} \in \mathcal{BV}^n$

Availability Specification for T

- Specification: $\mathcal{S}^{av,T,bvv}_G = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]^T_{av,bvv}, \vec{bv}_{\mathbf{s}})$

Variant 3: Fulfilling the Proof Obligations

Lemma 7.10.1.12 (Chain Conditions)

$\widehat{\mathbb{B}^n}$ and $\widehat{\mathcal{BV}^n}$ satisfy the descending and ascending chain condition.

Lemma 7.10.1.13 (Distributivity, Additivity)

$[\![\ ]\!]^T_{av,cpv}$ and $[\![\ ]\!]^T_{av,bvv}$ are distributive and additive.

Corollary 7.10.1.14 (Monotonicity)

$[\![\ ]\!]^T_{av,cpv}$ and $[\![\ ]\!]^T_{av,bvv}$ are monotonic.

Variant 3: Collecting the Guarantees (1)

...on termination.

Theorem 7.10.1.15 (Termination)

Applied to $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]_{av,cpv}^T, \bar{b}_{\mathsf{s}})$ or $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]_{av,bvv}^T, \vec{bv}_{\mathsf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/ *MinFP* semantics of $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$.

Proof. Immediately with Lemma 7.10.1.12, Corollary 7.10.1.14, and Termination Theorem 7.6.2.1.

Variant 3: Collecting the Guarantees (2)

...on tightness.

Theorem 7.10.1.16 (Tightness)

Applied to $\mathcal{S}_G^{av,T,cpv} = (\widehat{\mathbb{B}^n}, [\![\ ]\!]_{av,cpv}^T, \bar{b}_{\mathbf{s}})$ or $\mathcal{S}_G^{av,T,bvv} = (\widehat{\mathcal{BV}^n}, [\![\ ]\!]_{av,bvv}^T, \vec{bv}_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/ *JOP* tight for $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$, respectively (i.e., it terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,T,cpv}$ or $\mathcal{S}_G^{av,T,bvv}$, respectively).

Proof. Immediately with Lemma 7.10.1.13, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

Note:

- Applied to $\mathcal{S}_G^{av,T,bvv}$ instead of its cross-product counterpart, Algorithm 7.6.1.1 can take advantage of the efficient bitvector operations available on actual processors.
- This gives rise to call availability a bitvector problem.

Variant 4: Fixing the Setting

...availability for a finite set of terms T.

Introducing Gen/Kill Predicates for edges

- $$Gen_e^T =_{df} \{t \in T \mid Comp_e^t \wedge \neg Mod_e^t\} = \{t \in T \mid Comp_e^t \wedge Transp_e^t\}$$
- $$Kill_e^T =_{df} \{t \in T \mid Mod_e^t\} = \{t \in T \mid \neg Transp_e^t\}$$

Variant 4: Specifying the DFA (gen/kill view)

DFA Specification (gen/kill view (gkv))

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df} (\mathcal{P}(T), \cap, \cup, \subseteq, \emptyset, T) = \widehat{\mathcal{P}(T)}$
- DFA semantics
  $[\![\ ]\!]^{T}_{av,gkv} : E \rightarrow (\mathcal{P}(T) \rightarrow \mathcal{P}(T))$ where
  $\forall e \in E\ \forall T' \in \mathcal{P}(T).\ [\![\ e\ ]\!]^{T}_{av,gkv}(T') =_{df} (T' \backslash Kill^{T}_{e}) \cup Gen^{T}_{e}$
- Start assertion: $T_{\mathbf{s}} \in \mathcal{P}(T)$

Availability Specification for T

- Specification: $\mathcal{S}^{av,T,gkv}_{G} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]^{T}_{av,gkv}, T_{\mathbf{s}})$

Variant 4: Fulfilling the Proof Obligations

Comparing

- $[\![\ ]\!]^T_{av,gkv} : E \rightarrow (\,\mathcal{P}(T) \rightarrow \mathcal{P}(T)\,)$ where
  $\forall e \in E\, \forall T' \in \mathcal{P}(T).\, [\![\, e\,]\!]^T_{av,gkv}(T') =_{df} (\,T' \backslash Kill^T_e) \cup Gen^T_e$

with

- $[\![\ ]\!]^T_{av} : E \rightarrow (\,\mathcal{P}(T) \rightarrow \mathcal{P}(T)\,)$ where
  $$\forall\, e \in E\ \forall\, T' \in \mathcal{P}(T).\ [\![\, e\,]\!]^T_{av}(T') =_{df}$$
  $$\{t \in T \mid (t \in T' \vee Comp^t_e) \wedge Transp^t_e\}$$

...we get:

Lemma 7.10.1.17 (Equality)

$$[\![\ ]\!]^T_{av} = [\![\ ]\!]^T_{av,gkv}$$

Variant 4: Collecting the Guarantees

...on termination, tightness.

Theorem 7.10.1.18 (Termination)

Applied to $\mathcal{S}_G^{av,T,gkv} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av,gkv}^T, T_s)$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{av,T,gkv}$.

Proof. Immediately with Lemma 7.10.1.7, Lemma 7.10.1.8, Corollary 7.10.1.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.1.19 (Tightness)

Applied to $\mathcal{S}_G^{av,T,gkv} = (\widehat{\mathcal{P}(T)}, [\![\ ]\!]_{av,gkv}^T, T_s)$, Algorithm 7.6.1.1 is *MOP*/*JOP* tight for $\mathcal{S}_G^{av,T,gkv}$ (i.e., it terminates with the *MOP*/*JOP* semantics of $\mathcal{S}_G^{av,T,gkv}$).

Proof. Immediately with Lemma 7.10.1.7, Lemma 7.10.1.8, Coincidence Theorem 6.7.2, and Termination Theorem 7.6.2.1.

Availability again as a Gen/Kill-Problem (1)

...specializing the generic *MaxFP* Equation System 7.5.1.1:

Equation System 7.5.1.1 (*MaxFP* Equation System)

$$inf(n) = \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcap \{ [\![(m,n)]\!](inf(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

...for the availability problem yields:

Equation System 7.10.1.20 (Availability)

$$Available(n) =$$

$$\begin{cases} T_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigcap \{ [\![(m,n)]\!]^{T}_{av,gkv}(Available(m)) \mid m \in pred(n) \} & \text{otherwise} \end{cases}$$

Availability again as a Gen/Kill Problem (2)

...expanding additionally $[\![\ ]\!]^{T}_{av,gkv}$ we get:

Equation System 7.10.1.21 (Availability)

$Available(n) =$

$$\begin{cases} T_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigcap \{\, (Available(m) \backslash Kill^{T}_{(m,n)}) \cup Gen^{T}_{(m,n)} \mid m \in pred(n)\,\} & \text{otherwise} \end{cases}$$

Note: Both Equation System 7.10.1.21 and the definition of the DFA semantics

- $[\![\ ]\!]^{T}_{av,gkv} : E \rightarrow (\,\mathcal{P}(T) \rightarrow \mathcal{P}(T)\,)$ where
  $\forall e \in E\ \forall T' \in \mathcal{P}(T).\ [\![\, e\,]\!]^{T}_{av,gkv}(T') =_{df} (\,T' \backslash Kill^{T}_{e}) \cup Gen^{T}_{e}$

give rise to call availability a Gen/Kill problem.

Gen/Kill (or Bitvector) Problems

...including properties like

- availability and very busyness of terms, liveness and reaching definitions of variables, etc.

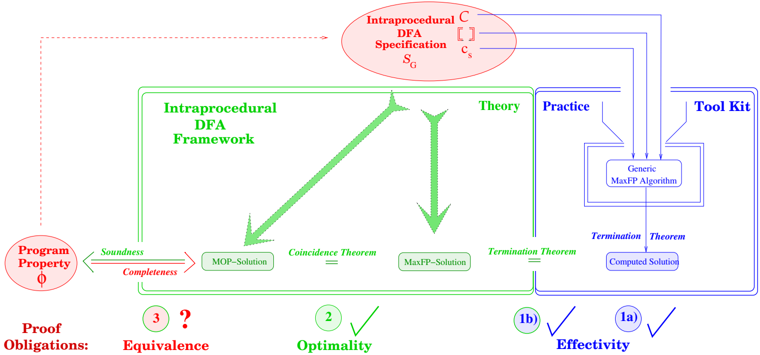
form despite their conceptual simplicity a most important class of DFA problems with numerous applications in program optimization including:

- Partially redundant expression elimination (busy/lazy code motion)
- Strength reduction (busy/lazy strength reduction)
- Partial dead-code elimination
- Partially redundant assignment elimination
- Assignment motion
- ...

...see course notes of LVA 185.A04 Optimizing Compilers for further details.

Variants 1 Thru 4: Closing the Final Proof Gap

...proving soundness and completeness for the *MOP* view of the availability property using $\mathcal{S}_G^{av,t}$ (Variant 1) as example:



Recall

...informally, a term is available at a node

- if, no matter which path is taken from the entry of the program to that node, the term is computed without that any of the variables occurring in it is redefined before reaching this node.

Note

- If entry of the program is replaced by entry of the procedure, the informal 'definition' of availibility does not foresee the possibility of the availability of an expression at the procedure entry itself.
- Situations where this availability is ensured by the calling context of the procedure, are thus not captured and can not be dealt with.

Towards defining Availability Formally

...useful notation.

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, and *Predicate* a predicate defined for edges $e \in E$.

For paths $p = \langle e_1, \ldots, e_q \rangle \in \mathbf{P}[m, n]$, we define:

- p_i, $1 \leq i \leq q$, denotes the i^{th} edge e_i of p.
- $p_{[k,l]}$ denotes the subpath $\langle e_k, \ldots, e_l \rangle$ of p.
- $\lambda_p = q$ denotes the length of p, i.e., the number of edges of p.

For predicates along paths, we define:

- $Predicate_p^{\forall} \iff_{df} \forall\, 1 \leq i \leq \lambda_p.\ Predicate_{p_i}$
- $Predicate_p^{\exists} \iff_{df} \exists\, 1 \leq i \leq \lambda_p.\ Predicate_{p_i}$

Availability

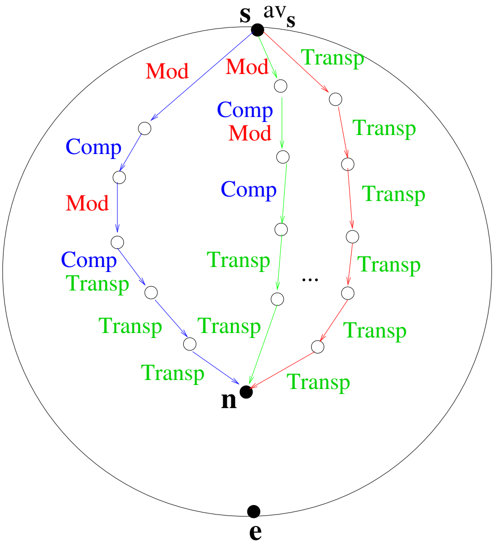
...defined formally:

Definition 7.10.1.22 (Availability)

Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, t a term, and $av_{\mathbf{s}} \in \mathbb{B}$ the availability information for t at $\mathbf{s}$ ensured by the calling context of G. Then:

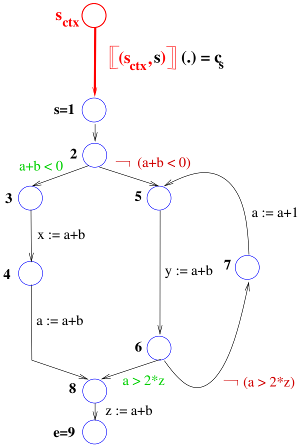
$$Available^t(n) \iff_{df}$$
$$\begin{cases} av_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \forall\, p \in \mathbf{P}[\mathbf{s}, n].\ (av_{\mathbf{s}}^t \wedge Transp^{t}{}^{\forall}_{p}) \ \vee & \\ \qquad\qquad \exists\, i \leq \lambda_p.\ Comp^t_{p_i} \wedge Transp^{t}{}^{\forall}_{p[i,\lambda_p]} & \text{otherwise} \end{cases}$$

Illustrating the Essence of Definition 7.10.1.22

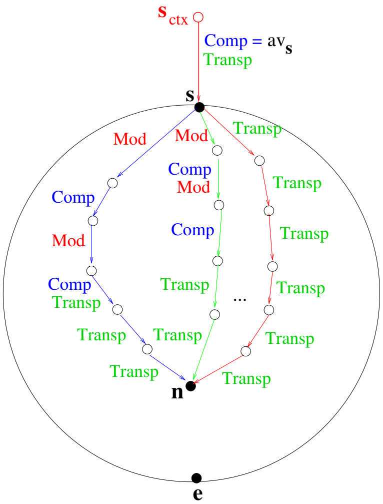


Context Edges

…allow a simpler case-free definition of availability:



Using Context Edges



...availability can be defined without cases:

$$\forall\, n \in N \backslash \{\mathbf{s}_{ctx}\}.\ Available^t(n) \iff_{df} \forall\, p \in \mathbf{P}[\mathbf{s}_{ctx}, n].\ \exists\, i \leq \lambda_p.\ Comp^t_{p_i} \wedge Transp^{t\,\forall}_{\ p[i,\lambda_p]}$$

Closing the Final Proof Gap

Theorem 7.10.1.23 (Soundness and Completeness)

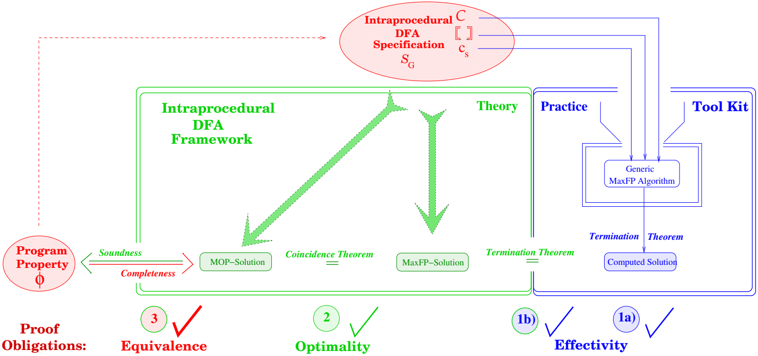
Let $G = (N, E, \mathbf{s}, \mathbf{e})$ be a flow graph, t an expression, $av_{\mathbf{s}} \in \mathbb{B}$ the availability information for t at $\mathbf{s}$ ensured by the calling context of G, and let $[\![\ ]\!]^{MOP}_{\mathcal{S}_G^{av,t}}$ be the *MOP* semantics of G for the DFA specification $\mathcal{S}_G^{av,t} = (\widehat{\mathbb{B}}, [\![\ ]\!]^t_{av}, av_{\mathbf{s}}, fw)$.

Then:

$$\forall\, n \in N.\ Available^t(n) \iff [\![\, n \,]\!]^{MOP}_{\mathcal{S}_G^{av,t}}$$

Gap Closed: Soundness and Completeness

...for the *MOP* view of $\mathcal{S}_G^{av,t}$ for term availability proven:



Homework: Exercise 7.10.1.24

1. What does soundness and completeness mean
2. How can soundness and completeness be proven

...for the *JOP* view of $\mathcal{S}_G^{av,t}$ for term availability?

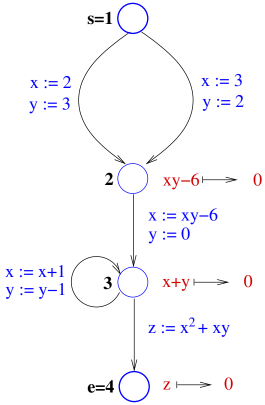
Chapter 7.10.2

Monotonic DFA: Simple Constants

Intuitively

...a term is a constant of value c at a node, if, no matter which path is taken from the entry of the program to that node, the evaluation of this term at the node yields value c.

Illustration:



Example by Markus Müller-Olm, Helmut Seidl (SAS 2002)

Constant Propagation and Folding

...terms of a constant value can be replaced at compile time by this value effectively moving computational effort from the run time of a program to its compile time improving its run time performance, a so-called program optimization known as constant propagation and folding.

Unfortunately, there is no algorithm which always succeeds in determining if a term is a constant of some value at a node or not.

Undecidability of Constant Propagation

Theorem 7.10.2.1 (Undecidability, Reif&Lewis 1977)

In the arithmetic domain, the problem of discovering all text expressions covered by constant signs is undecidable.

(John H. Reif, Harry R. Lewis. Symbolic Evaluation and the Global Value Graph. In Conference Record of the 4th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977)

Proof Sketch of Theorem 7.10.2.1 (1)

The proof of Theorem 7.10.2.1 works by reducing Hilbert's 10th problem to the problem of discovering all text expressions covered by constant signs:

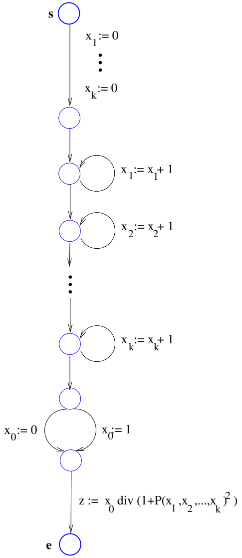
- Hilbert's 10th Problem

  Let $\{x_1, \ldots, x_k\}$ be a set of variables, $k > 5$, and let $P(x_1, \ldots, x_k)$ be a (multivariate) polynomial.

  It is not decidable, if $P(x_1, \ldots, x_k)$ has a root in the natural numbers (Matijasevic 1970).

Proof Sketch of Theorem 7.10.2.1 (2)

...consider program G given by its below flow graph:



Proof Sketch of Theorem 7.10.2.1 (3)

Then: Proving the equivalence

P has no root in the natural numbers iff

z is of a constant value at node **e** of G

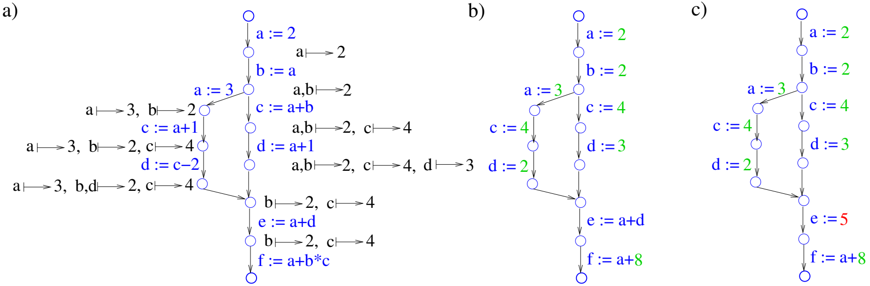
completes the proof. □

Simple Constants

...due to this negative result, in practice simpler decidable versions of the constant propagation and folding problem are considered, one of which is the class of so-called simple constants.

Informally, a term is a simple constant at a node, if every operand of the term has a unique constant value at this node, no matter, which path is taken from the entry of the program to the node.

Illustrating Simple Constants (1)



Note:

- All terms except of $a + d$ and $a + 8$ are simple constants (Figure b)).
- $a + d$ is a constant of value 5 but not a simple constant (Figure c)).

Illustrating Simple Constants (2)

- None of the (none-trivial) terms in the initial example of Müller-Olm and Seidl is a simple constant.
- $a + d$ as well as all terms in the example of Müller-Olm and Seidl can be detected to be constants by more sophisticated (and in the latter case computationally considerably more complex) constant propagation algorithms (cf. course notes of LVA 185.A04 Optimizing Compilers for details).

...computing simple constants is

- a canonical example of a monotonic (non distributive) DFA problem.
- an example of an incomplete analysis algorithm, which fails to dectect many terms to be constant, which could be detected so, but which is efficient w/ still useful results for optimization: Trading completeness for efficiency!

Computing Simple Constants: Preliminaries

...from data domains to DFA lattices.

Let $\mathbb{D}$ be the

- data domain of interest (e.g., the set of natural numbers $\mathbb{N}$, the set of integers $\mathbb{Z}$, the set of Boolean truth values $\mathbb{B}$, etc.) with a distinguished element $\perp$ representing the value *undefined*.

We extend $\mathbb{D}$ by adding

- a new element $\top$ not in $\mathbb{D}$, i.e., $\top \notin \mathbb{D}$

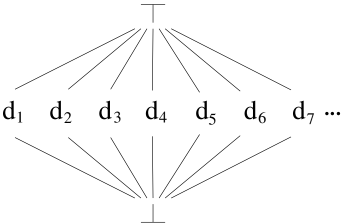
...and denote the extended domain by

- $\mathbb{D}' =_{df} \mathbb{D} \cup \{\top\}$.

Note: Assuming $\perp$ an element of the underlying data domain, whereas $\top$ not, might appear arbitrary. It is motivated by the fact that data types implemented on machines often contain a representation considered the undefined value of the data type.

Constructing DFA Lattices

...given an extended data domain $\mathbb{ID}'$, we construct the flat lattice $\mathcal{FL}_{\mathbb{ID}'}$ (cf. Appendix A.4)



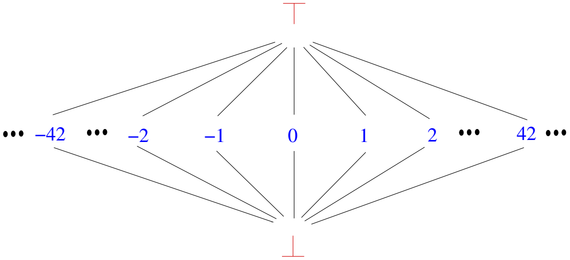
which is the basic DFA lattice of the DFA analysis for simple constants.

Intuitively

- $\top$ represents complete but inconsistent information.
- d_i, $i \geq 1$, represents accurate information.
- $\perp$ represents no information, the 'empty' information.

The Basic DFA Lattice over $\mathbb{Z}$

...is given by $\mathcal{FL}_{\mathbb{Z}}$:



...which is used for computing the class of simple constants over $\mathbb{Z}$.

Abstract Program States: DFA States

Definition 7.10.2.2 (DFA States)

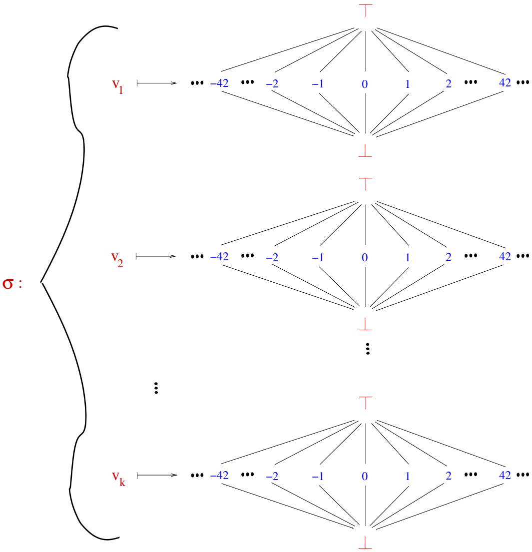
1. A DFA state is a total mapping $\sigma : \mathbf{V} \to \mathrm{ID}'$, which maps every variable to a datum $d \in \mathrm{ID}'$.
2. The set of all DFA states is defined by

$$\Sigma' =_{df} \{\, \sigma \mid \sigma : \mathbf{V} \to \mathrm{ID}' \,\}.$$

3. $\sigma_\perp$ and $\sigma_\top$ denote two distinguished DFA states of Σ', which are defined by:

$$\forall\, v \in \mathbf{V}.\ \sigma_\perp(v) = \perp,\ \sigma_\top(v) = \top.$$

Illustrating a DFA State σ over $\mathbb{Z}$



Initial DFA States

...for an initial DFA state, we require that no variable is mapped to the special value $\top$, i.e, we require to either have accurate information of the value of a variable, when entering a procedure, or no information at all. We define:

Definition 7.10.2.3 (Initial DFA States over ID′)

The set of initial DFA states is defined by

$$\Sigma'_{Init} =_{df} \{ \sigma \in \Sigma' \mid \forall\, v \in \mathbf{V}.\ \sigma(v) \neq \top \}$$

Extending the Interpretation

...of constant and operator symbols from $\mathbb{ID}$ to $\mathbb{ID}'$.

Definition 7.10.2.4 (Extending the Interpretation)

Let $I =_{df} (\ \mathbb{ID}, I_0\ )$ be an interpretation of constant and operator symbols over the data domain $\mathbb{ID}$.

Then $I' =_{df} (\ \mathbb{ID}', I_0'\ )$ is an interpretation over $\mathbb{ID}'$ which extends I by defining

- $I_0'(c) =_{df} I_0(c)$ for every constant symbol $c \in \mathbf{C}$
- $I_0'(op) : \mathbb{ID}'^k \to \mathbb{ID}'$ for every k-ary operator symbol $op \in \mathbf{O}$:

$$\forall\,(d_1, \ldots, d_k) \in \mathbb{ID}'^k.\ I_0'(op)(d_1, \ldots, d_k) =_{df}$$

$$\begin{cases} I_0(op)(d_1, \ldots, d_k) & \text{if } d_i = \bot \text{ for some } 1 \le i \le k \text{, or} \\ & \quad d_j \neq \top,\ 1 \le j \le k \\ \top & \text{if } d_i \neq \bot,\ 1 \le i \le k \text{, and} \\ & \quad d_j = \top \text{ for some } 1 \le j \le k \end{cases}$$

The Abstract Term Semantics over $\mathbb{D}'$

Definition 7.10.2.5 (Abstract Term Semantics)

The abstract semantics of terms $t \in \mathbf{T}$ is defined by the evaluation function

$$\mathcal{E}\colon \mathbf{T} \to (\Sigma' \to \mathbb{D}')$$

defined by

$$\forall t \in \mathbf{T}\ \forall \sigma \in \Sigma'.\ \mathcal{E}(t)(\sigma) =_{df} \begin{cases} \sigma(x) & \text{if } t \equiv x \in \mathbf{V} \\ I'_0(c) & \text{if } t \equiv c \in \mathbf{C} \\ I'_0(op)(\mathcal{E}(t_1)(\sigma), \ldots, \mathcal{E}(t_k)(\sigma)) & \\ \qquad\qquad \text{if } t \equiv (op, t_1, \ldots, t_k) & \end{cases}$$

The Abstract Instruction Semantics

Definition 7.10.2.6 (Abstract Instruction Semantics)

The abstract semantics of

- an assignment instruction $\iota \equiv x := t$ is given by the state transformer $\theta_\iota \colon \Sigma' \to \Sigma'$ defined by

$$\forall\, \sigma \in \Sigma'\ \forall\, y \in \mathbf{V}.\ \theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

- the empty instruction $\iota \equiv \mathit{skip}$ and a condition $\iota \equiv \mathit{cond}$ is given by the identical state transformer $Id_{\Sigma'}$, i.e., $\theta_\iota =_{df} Id_{\Sigma'}$ with $Id_{\Sigma'} \colon \Sigma' \to \Sigma'$ defined by $\forall\, \sigma \in \Sigma'.\ Id_{\Sigma'}(\sigma) =_{df} \sigma$.

Note: Executing *skip* and evaluating conditions do not have side effects.

The DFA Lattice for Simple Constants

...the set of DFA states together with the pointwise ordering of states, $\sqsubseteq_{\Sigma'}$, forms a complete lattice (cf. Appendix A.4):

$$\forall\, \sigma, \sigma' \in \Sigma'.\ \sigma \sqsubseteq_{\Sigma'} \sigma' \ \text{ iff } \ \forall\, v \in \mathbf{V}.\ \sigma(v) \sqsubseteq_{\mathcal{FL}_{\mathbb{D}'}} \sigma'(v)$$

Lemma 7.10.2.7 (Lattice of DFA States)

$\widehat{\Sigma'} =_{df} (\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\perp, \sigma_\top)$ is a complete lattice with

- least element $\sigma_\perp$,
- greatest element $\sigma_\top$,
- pointwise meet $\sqcap_{\Sigma'}$ and join $\sqcup_{\Sigma'}$ as meet and join operation, respectively.

Simple Constants: Specifying the DFA

DFA Specification

- DFA lattice
  $\widehat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \bot, \top) =_{df}$
  $$(\Sigma', \sqcap_{\Sigma'}, \sqcup_{\Sigma'}, \sqsubseteq_{\Sigma'}, \sigma_\bot, \sigma_\top) = \widehat{\Sigma}'$$
  with Σ' set of DFA states over $\mathbb{Z}$.
- DFA semantics
  $[\![\ ]\!]_{sc} : E \to (\Sigma' \to \Sigma')$ where $\forall\, e \in E.\ [\![e]\!]_{sc} =_{df} \theta'_{\iota_e}$
- Start assertion: $\sigma_{\mathbf{s}} \in \Sigma'_{Init}$

Simple Constants Specification

- Specification: $\mathcal{S}_G^{sc} = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$

Simple Constants: Fulfilling the Proof Oblig.

Lemma 7.10.2.8 (Chain Conditions)

$\widehat{\Sigma}'$ satisfies the descending and ascending chain condition.

Note: The set of variables occurring in a program is finite.

Lemma 7.10.2.9 (Monotonicity)

$[\![\ ]\!]_{sc}$ is monotonic.

Lemma 7.10.2.10 (Non-Distributivity/Addititivity)

$[\![\ ]\!]_{sc}$ is (in general) not distributive and not additive.

Simple Constants: Collecting the Guarantees

...on termination, conservativity.

Theorem 7.10.2.11 (Termination)

Applied to $\mathcal{S}_G^{sc} = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 terminates with the *MaxFP*/*MinFP* semantics of $\mathcal{S}_G^{sc}$.

Proof. Immediately with Lemma 7.10.2.8, Lemma 7.10.2.9, and Termination Theorem 7.6.2.1.

Theorem 7.10.2.12 (Safety, Conservativity)

Applied to $\mathcal{S}_G^{sc} = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 is *MOP*/*JOP* conservative for $\mathcal{S}_G^{sc}$ (i.e., it terminates with a lower (upper) approximation of the *MOP*/*JOP* semantics of $\mathcal{S}_G^{sc}$, resp.).

Proof. Immediately with Lemma 7.10.2.8, Lemma 7.10.2.9, Safety Theorem 7.7.1, and Termination Theorem 7.6.2.1.

Simple Constants: Negative Result

...on tightness.

Theorem 7.10.2.13 (Non-Tightness)

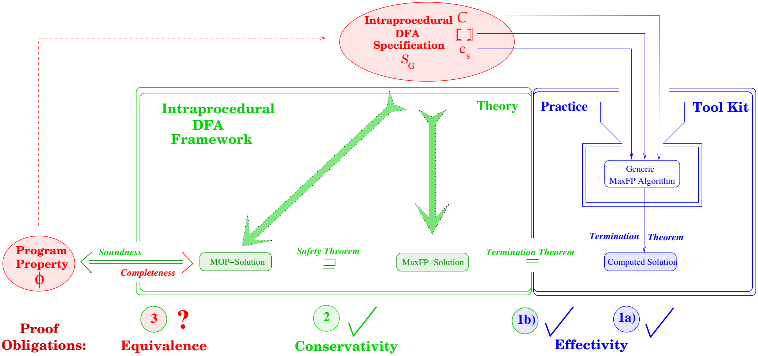
Applied to $\mathcal{S}_G^{sc} = (\widehat{\Sigma}', [\![\ ]\!]_{sc}, \sigma_{\mathbf{s}})$, Algorithm 7.6.1.1 is in general not *MOP*/*JOP* tight for $\mathcal{S}_G^{sc}$ (i.e., it terminates with a proper approximation of the *MOP*/*JOP* solution of $\mathcal{S}_G^{sc}$, respectively).

Proof. Immediately with Lemma 7.10.2.8, Lemma 7.10.2.9, Lemma 7.10.2.10, Coincidence Theorem 7.7.2, and Termination Theorem 7.6.2.1.

In closing: The *MaxFP*/*MinFP* solutions of $\mathcal{S}_G^{sc}$ are always safe approximations of the *MOP*/*JOP* solutions of $\mathcal{S}_G^{sc}$. In general, the operational *MOP*/*JOP* solutions of $\mathcal{S}_G^{sc}$ and their denotational *MaxFP*/*MinFP* counterparts do not coincide.

Simple Constants: Closing the Final Proof Gap

...proving soundness and completeness for the *MOP* view of $\mathcal{S}_G^{sc}$ for the simple constant property:



Simple Constants: Soundness, Completeness

...for the *MOP* semantics.

Theorem 7.10.2.14 (Soundness and Completeness)

The *MOP* semantics of $\mathcal{S}_G^{sc}$ is

1. sound and complete for variables.
2. sound but not complete for (non-trivial) terms (i.e., for terms containing at least one (non-unary) operator symbol).

...for Theorem 7.10.2.14(2), note that the *MOP* solution at every node can be considered a state, i.e., a map from variables to values, allowing an evaluation of terms.

Simple Constants: Soundness, Completeness

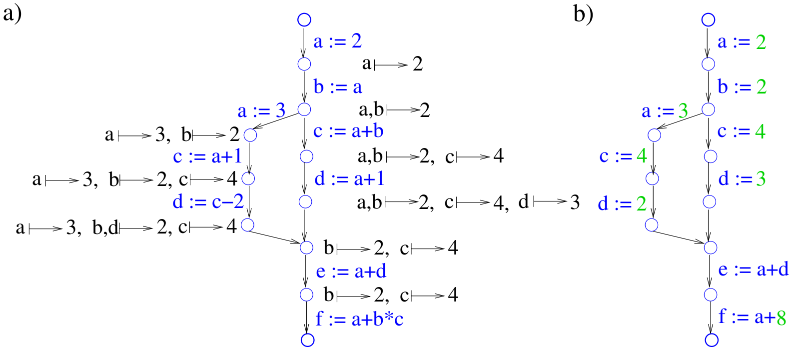
...for the *MaxFP* semantics.

Theorem 7.10.2.15 (Soundness and Completeness)

The *MaxFP* semantics of $\mathcal{S}_G^{sc}$ is sound but not complete (for both variables and terms).

...see course notes of LVA 185.A04 Optimizing Compilers for further details.

Simple Constants: Illustrating Example

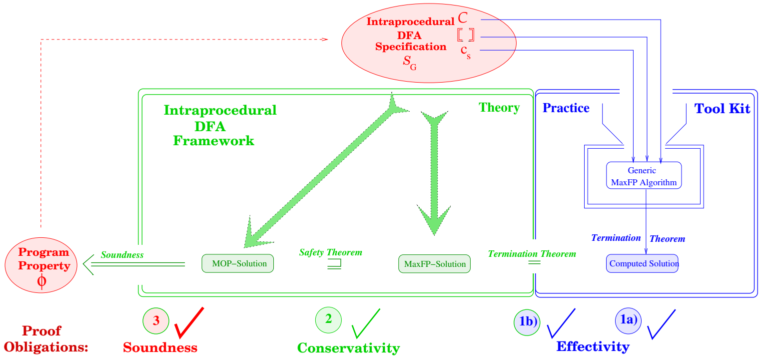


...all terms except of $a + d$ and $a + 8$ are simple constants.

Recall: $a + d$ is a constant of value 5 but not a simple constant; $a + 8$ is not a constant.

Gap Partially Closed: Soundness

...for the *MOP* view of $\mathcal{S}_G^{sc}$ for simple constants proven:



Homework: Exercise 7.10.2.16

1. What does soundness and completeness mean
2. How can soundness and completeness be proven

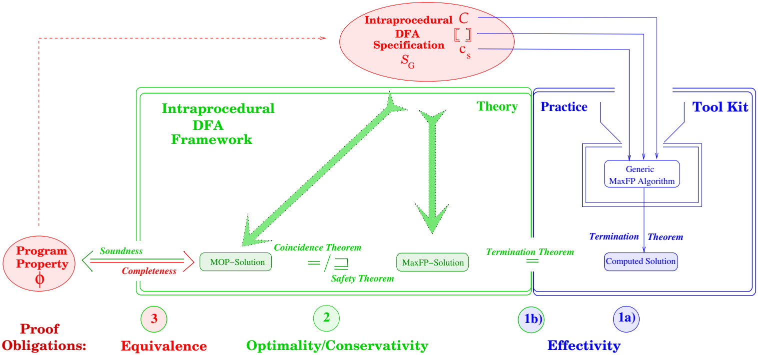
...for the *JOP* view of $\mathcal{S}_G^{sc}$ for the simple constants property?

Chapter 7.11

Summary, Looking Ahead

The Framework/Toolkit View

...of data flow analysis.



...reconsidered from the angle of correctness, accuracy, and the kind of ϕ.

Kinds of Properties: Must vs. May

...basically, we can distinguish two kinds of properties ϕ:

- Universally quantified (or must) properties $\phi^\forall$: $\phi^\forall$ holds at a node n, if it holds along all paths from **s** to n at n.
- Existentially quantified (or may) properties $\phi^\exists$: $\phi^\exists$ holds at a node n, if it holds along some paths from **s** to n at n.

Must-properties $\phi^\forall$ are related to the

- operational *MOP* semantics of a program and its computational denotational counterpart, the *MaxFP* semantics.

May-properties $\phi^\exists$ are related to the

- operational *JOP* semantics of a program and its computational denotational counterpart, the *MinFP* semantics.

Correctness and Accuracy

...essentially, there are two places where correctness and accuracy issues are handled in the framework/toolkit view of DFA:

Framework/Toolkit **internally**: captured by

- **Safety** $\rightsquigarrow$ Conservativity
- **Coincidence** $\rightsquigarrow$ Tightness

...relating *MaxFP*/*MinFP* and *MOP*/*JOP* solution, respectively.

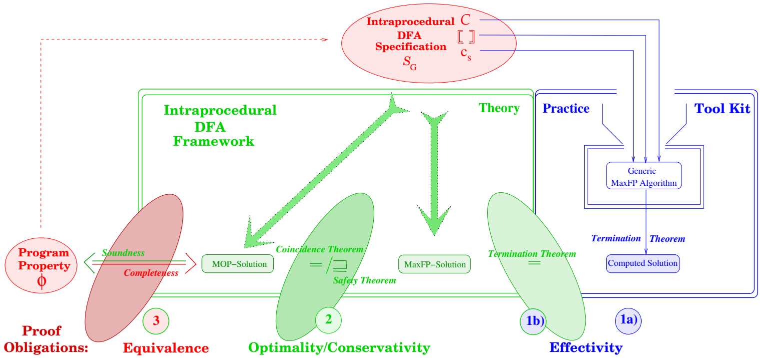
Framework/Toolkit **externally**: captured by

- **Soundness** $\rightsquigarrow$ No false positives
- **Completeness** $\rightsquigarrow$ No false negatives

...relating *MOP*/*JOP* solution and $\phi^\forall/\phi^\exists$, respectively.

Illustrating

...the places of internal and external correctness and accuracy handling:

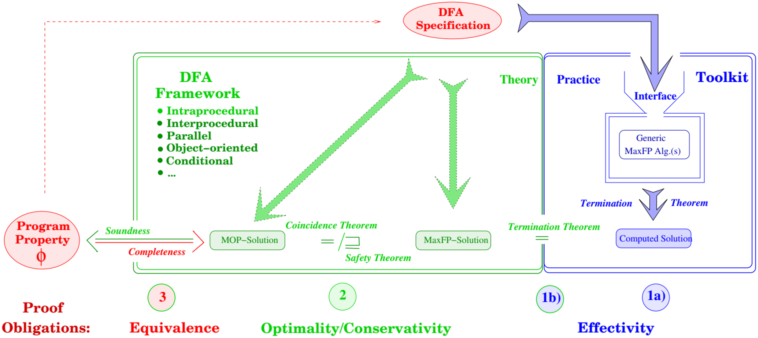


Looking ahead: The Uniform View of DFA

...in the course of this lecture course (and of LVA 185.A04 Optimizing Compilers), we will see:

- The Framework and Toolkit View of DFA

is achievable beyond the base case of intraprocedural DFA providing a uniform view of DFA:



Chapter 7.12

References, Further Reading

Further Reading for Chapter 7 (1)

Textbook Representations

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007. (Chapter 1.2, The Structure of a Compiler; Chapter 1.4, The Science of Building a Compiler; Chapter 1.4.2, The Science of Code Optimization; Chapter 9.1, The Principal Sources of Program Optimization)
- Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Appendix B.3.1, Graphical Intermediate Representations)
- Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.

Further Reading for Chapter 7 (2)

- Uday P. Khedker, Amitabha Sanyal, Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. CRC Press, 2009. (Chapter 3, Theoretical Abstractions in Data Flow Analysis; Chapter 4, General Data Flow Frameworks; Chapter 5, Complexity of Iterative Data Flow Analysis)
- Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998. (Chapter 2.3, Building the Flow Graph; Chapter 4.7, Structure of Program Flow Graph)
- Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Chapter 7, Control-Flow Analysis)

Further Reading for Chapter 7 (3)

- Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)
- Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)

Further Reading for Chapter 7 (4)

Pioneering, Groundbreaking Articles

- Frances E. Allen, John A. Cocke. *A Program Data Flow Analysis Procedure*. Communications of the ACM 19(3):137-147, 1976.
- Susan Horwitz, Alan J. Demers, Tim Teitelbaum. *An Efficient General Iterative Algorithm for Dataflow Analysis*. Acta Informatica 24(6):679-694, 1987.
- Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.
- John B. Kam, Jeffrey D. Ullman. *Global Data Flow Analysis and Iterative Algorithms*. Journal of the ACM 23:158-171, 1976.

Further Reading for Chapter 7 (5)

- John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.

Frameworks and Toolkits

- Marion Klein, Jens Knoop, Dirk Koschützki, Bernhard Steffen. *DFA&OPT-METAFrame: A Toolkit for Program Analysis and Optimization*. In Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), Springer-V., LNCS 1055, 422-426, 1996.

- Jens Knoop. *From DFA-Frameworks to DFA-Generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.

Further Reading for Chapter 7 (6)

- Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 28(2):121-163, 1990.
- Stephen P. Masticola, Thomas J. Marlowe, Barbara G. Ryder. *Lattice Frameworks for Multisource and Bidirectional Data Flow Problems*. ACM Transactions on Programming Languages and Systems (TOPLAS) 17(5):777-803, 1995.
- Florian Martin. *PAG - An Efficient Program Analyzer Generator*. Journal of Software Tools for Technology Transfer 2(1):46-67, 1998.
- Flemming Nielson. *Semantics-directed Program Analysis: A Tool-maker's Perspective*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 2-21, 1996.

Further Reading for Chapter 7 (7)

Solving Equation Systems, Computing Fixed Points

- Christian Fecht, Helmut Seidl. *An Even Faster Solver for General Systems of Equations*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 189-204, 1996.
- Christian Fecht, Helmut Seidl. *Propagating Differences: An Efficient New Fixpoint Algorithm for Distributive Constraint Systems*. In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 90-104, 1998.
- Christian Fecht, Helmut Seidl. *A Faster Solver for General Systems of Equations*. Science of Computer Programming 35(2):137-161, 1999.

Further Reading for Chapter 7 (8)

Bernhard Steffen, Andreas Claßen, Marion Klein, Jens Knoop, Tiziana Margaria. *The Fixpoint Analysis Machine*. In Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95), Springer-V., LNCS 962, 72-87, 1995.

Flow Graph Pragmatics

Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.

Further Reading for Chapter 7 (9)

- Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65-79, 1998.

Miscellaneous

- Stephen M. Blackburn, Amer Diwan, Matthias Hauswirth, Peter F. Sweeny, José Nelson Amaral, Tim Brecht, Lubomír Bulej, Cliff Click, Lieven Eeckhout, Sebastian Fischmeister, Daniel Frampton, Laurie J. Hendren, Michael Hind, Antony L. Hosking, Richard E. Jones, Tomas Kalibera, Nathan Keynes, Nathaniel Nystrom, Andreas Zeller. *The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations*. ACM Transactions on Programming Languages and Systems 38(4), Article 15:1-20, 2016.

Further Reading for Chapter 7 (10)

- Martin Davis. *Hilbert's Tenth Problem is Unsolvable*. American Mathematical Monthly 80:33-269, 1973.
- Martin Davis, Yuri Matijasevič, Julia Robinson. *Hilbert's Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution*. In Proceedings of the Symposium on the Hilbert Problems (De Kalb, Illinois), May 1974, American Mathematical Society, Providence, R.I., 323-378, 1976.
- William Landi. *Undecidability of Static Analysis*. ACM Letters on Programming Languages and Systems 1(4):323-337, 1992.

Further Reading for Chapter 7 (11)

- Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 7, What can one tell about a Program without its Execution: Static Analysis)
- Yuri V. Matijasevic. *Enumerable Sets are Diophantine (In Russian)*. Dodl. Akad. Nauk SSSR 191, 279-282, 1970.
- Yuri V. Matijasevič. *On Recursive Unsolvability of Hilbert's Tenth Problem*. In Proceedings of the 4th International Congress on Logic, Methodology and Philosophy of Science (Bucharest 1971), North-Holland, Amsterdam, 89-110, 1973.
- Yuri V. Matijasevic. *What Should We Do Having Proved a Decision Problem to be Unsolvable?* Algorithms in Modern Mathematics and Computer Science 1979:441-448, 1979.

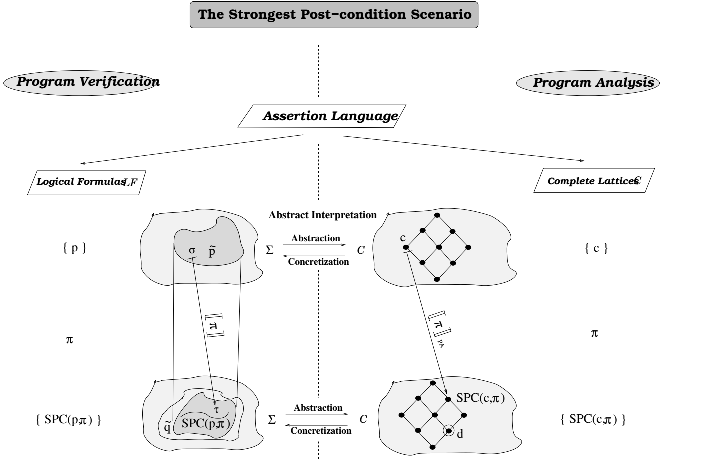
Further Reading for Chapter 7 (12)

- Yuri V. Matijasevic. *Hilbert's Tenth Problem*. MIT Press, 1993.
- Markus Müller-Olm, Helmut Seidl. *Polynomial Constants are Decidable*. In Proceedings of the 9th Static Analysis Symposium (SAS 2002), Springer-V., LNCS 2477, 4-19, 2002.

Chapter 10

Program Verification vs. Program Analysis: Axiomatic Verification, Data Flow Analysis Compared

Strongest Post-Condition View in PV and PA



$SPC(p,\pi) \in LF$ *must satisfy:*

(1) $\models_{PV} \{ p \}\pi\{ SPC(p,\pi) \}$

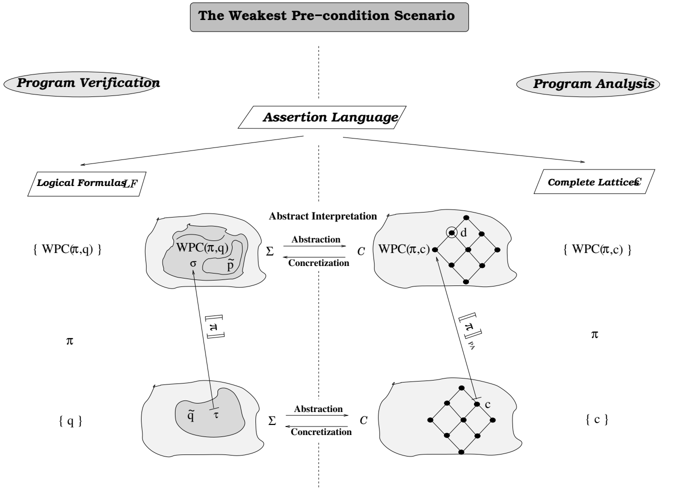
(2) $\forall\, q \in LF.\ \models_{PV} \{ p \}\pi\{ q \}$ implies $SPC(p,\pi) \Longrightarrow q$

$SPC(c,\pi) \in C$ *must satisfy:*

(1) $\models_{PA} \{ c \}\pi\{ SPC(c,\pi) \}$

(2) $\forall\, d \in C.\ \models_{PA} \{ c \}\pi\{ d \}$ implies $SPC(c,\pi) \sqsubseteq d$

Weakest Pre-Condition View in PV and PA



$WPC(\pi,q) \in LF$ *must satisfy:*

(1) $\models_{PV} \{\, WPC(\pi,q) \,\}\, \pi \,\{\, q \,\}$

(2) $\forall\, p \in LF.\ \models_{PV} \{\, p \,\}\pi\{\, q \,\}$ implies $p \Longrightarrow WPC(\pi,q)$

$WPC(\pi,c) \in C$ *must satisfy:*

(1) $\models_{PA} \{\, WPC(\pi,c) \,\}\, \pi \,\{\, c \,\}$

(2) $\forall\, d \in C.\ \models_{PA} \{\, d \,\}\pi\{\, c \,\}$ implies $d \sqsupseteq WPC(\pi,c)$

Appendices

Appendix A

Mathematical Foundations

A.1

Relations

Relations

Let M_i, $1 \leq i \leq k$, be sets.

Definition A.1.1 (k-ary Relation)

A (k-ary) relation is a set R of ordered tuples of elements of $M_1, \ldots, M_k$, i.e., $R \subseteq M_1 \times \ldots \times M_k$ is a subset of the cartesian product of the sets M_i, $1 \leq i \leq k$.

Examples

- $\emptyset$ is the smallest relation on $M_1 \times \ldots \times M_k$.
- $M_1 \times \ldots \times M_k$ is the biggest relation on $M_1 \times \ldots \times M_k$.

Binary Relations

Let M, N be sets.

Definition A.1.2 (Binary Relation)

A (binary) relation is a set R of ordered pairs of elements of M and N, i.e., R is a subset of the cartesian product of M and N, $R \subseteq M \times N$, called a relation from M to N.

Examples

- $\emptyset$ is the smallest relation from M to N.
- $M \times N$ is the biggest relation from M to N.

Note

- If R is a relation from M to N, it is common to write $m\, R\, n$, $R(m, n)$, or $R\, m\, n$ instead of $(m, n) \in R$.

Between, On

Definition A.1.3 (Between, On)

A relation R from M to N is called a relation between M and N (or a relation on $M \times N$).

If M equals N, then R is called a relation on M, in symbols: (M, R).

Domain and Range of a Binary Relation

Definition A.1.4 (Domain and Range)

Let R be a relation from M to N.

The sets

- $dom(R) =_{df} \{m \mid \exists\, n \in N.\ (m,n) \in R\}$
- $ran(R) =_{df} \{n \mid \exists\, m \in M.\ (m,n) \in R\}$

are called the domain and the range of R, respectively.

Properties of Relations on a Set M

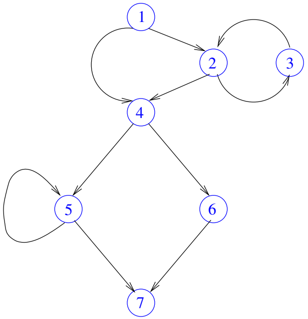
Definition A.1.5 (Properties of Relations on M)

A relation R on a set M is called

- reflexive iff $\forall\, m \in M.\; m\, R\, m$
- irreflexive iff $\forall\, m \in M.\; \neg m\, R\, m$
- transitive iff $\forall\, m, n, p \in M.\; m\, R\, n \;\wedge\; n\, R\, p \;\Rightarrow\; m\, R\, p$
- intransitive iff $\forall\, m, n, p \in M.\; m\, R\, n \wedge n\, R\, p \Rightarrow \neg m\, R\, p$
- symmetric iff $\forall\, m, n \in M.\; m\, R\, n \iff n\, R\, m$
- antisymmetric iff $\forall\, m, n \in M.\; m\, R\, n \wedge n\, R\, m \Rightarrow m = n$
- asymmetric iff $\forall\, m, n \in M.\; m\, R\, n \Rightarrow \neg n\, R\, m$
- linear iff $\forall\, m, n \in M.\; m\, R\, n \vee n\, R\, m \vee m = n$
- total iff $\forall\, m, n \in M.\; m\, R\, n \vee n\, R\, m$

(Anti-) Example

Let $G = (N, E, \mathbf{s} \equiv 1, \mathbf{e} \equiv 7)$ be the below (flow) graph, and let R be the relation '$\cdot$ is linked to $\cdot$ via a (directed) edge' on N of G (e.g., node 4 is linked to node 6 but not vice versa).



The relation R is not reflexive, not irreflexive, not transitive, not intransive, not symmetric, not antisymmetric, not asymmetric, not linear, and not total.

Equivalence Relation

Let R be a relation on M.

Definition A.1.6 (Equivalence Relation)

R is an equivalence relation (or equivalence) iff R is reflexive, transitive, and symmetric.

Exercise A.1.7

Let $\mid$ denote the divisibility relation on the set of natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \mid 35$.

Prove or disprove: The divisibility relation $\mid$ on $\mathbb{N}_0$ is

1. reflexive
2. irreflexive
3. transitive
4. intransitive
5. symmetric
6. antisymmetric
7. asymmetric
8. linear
9. total
10. equivalence (relation)

Proof or counterexample.

A.2

Ordered Sets

A.2.1

Pre-Orders, Partial Orders, and More

Ordered Sets

Let R be a relation on M.

Definition A.2.1.1 (Pre-Order)

R is a pre-order (or quasi-order) iff R is reflexive and transitive.

Definition A.2.1.2 (Partial Order)

R is a partial order (or poset or order) iff R is reflexive, transitive, and antisymmetric.

Definition A.2.1.3 (Strict Partial Order)

R is a strict partial order iff R is asymmetric and transitive.

Examples of Ordered Sets

Pre-order (reflexive, transitive)

- The relation $\Rightarrow$ on logical formulas.

Partial order (reflexive, transitive, antisymmetric)

- The relations $=$, $\leq$ and $\geq$ on $\mathbb{N}$.
- The relation $m \mid n$ (m is a divisor of n) on $\mathbb{N}$.

Strict partial order (asymmetric, transitive)

- The relations $<$ and $>$ on $\mathbb{N}$.
- The relations $\subset$ and $\supset$ on sets.

Equivalence relation (reflexive, transitive, symmetric)

- The relation $\iff$ on logical formulas.
- The relation 'have the same prime number divisors' on $\mathbb{N}$.
- The relation 'are citizens of the same country' on people.

Note

- An antisymmetric pre-order is a partial order; a symmetric pre-order is an equivalence relation.
- For convenience, also the pair (M, R) is called a pre-order, partial order, and strict partial order, respectively.
- More accurately, we could speak of the pair (M, R) as of a set M which is pre-ordered, partially ordered, and strictly partially ordered by R, respectively.
- Synonymously, we also speak of M as a pre-ordered, partially ordered, and a strictly partially ordered set, respectively, or of M as a set which is equipped with a pre-order, partial order and strict partial order, respectively.
- On any set, the equality relation $=$ is a partial order, called the discrete (partial) order.

The Strict Part of an Ordering

Let $\sqsubseteq$ be a pre-order (reflexive, transitive) on P.

Definition A.2.1.4 (Strict Part of $\sqsubseteq$)

The relation $\sqsubset$ on P defined by

$$\forall\, p, q \in P.\; p \sqsubset q \iff_{df} p \sqsubseteq q \;\wedge\; p \neq q$$

is called the strict part of $\sqsubseteq$.

Corollary A.2.1.5 (Strict Partial Order)

Let $(P, \sqsubseteq)$ be a partial order, let $\sqsubset$ be the strict part of $\sqsubseteq$.

Then: $(P, \sqsubset)$ is a strict partial order.

Useful Results

Let $\sqsubset$ be a strict partial order (asymmetric, transitive) on P.

Lemma A.2.1.6

The relation $\sqsubset$ is irreflexive.

Lemma A.2.1.7

The pair $(P, \sqsubseteq)$, where $\sqsubseteq$ is defined by

$$\forall\, p, q \in P.\ p \sqsubseteq q \iff_{df} p \sqsubset q \ \vee \ p = q$$

is a partial order.

Induced (or Inherited) Partial Order

Definition A.2.1.8 (Induced Partial Order)

Let $(P, \sqsubseteq_P)$ be a partially ordered set, let $Q \subseteq P$ be a subset of P, and let $\sqsubseteq_Q$ be the relation on Q defined by

$$\forall\, q, r \in Q.\ q \sqsubseteq_Q r \iff_{df} q \sqsubseteq_P r$$

Then: $\sqsubseteq_Q$ is called the induced partial order on Q (or the inherited order from P on Q).

Exercise A.2.1.9

Let $|$ denote the divisibility relation on the set of natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \mid 35$.

Prove or disprove: The divisibility relation $|$ on $\mathbb{N}_0$ is a

1. pre-order
2. partial order
3. strict partial order
4. equivalence (relation)

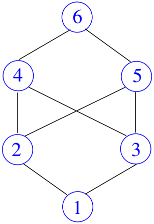
Proof or counterexample.

A.2.2

Hasse Diagrams

Hasse Diagrams

...are a sparse graphical representation of partial orders.



The links of a Hasse diagram

- are read from below to above (lower means smaller).
- represent the relation R of '$\cdot$ is an immediate predecessor of $\cdot$' defined by
  $p\,R\,q \iff_{df} \; p \sqsubset q \wedge \nexists\, r \in P.\; p \sqsubset r \sqsubset q$
  of a partial order $(P, \sqsubseteq)$, where $\sqsubset$ is the strict part of $\sqsubseteq$.

Reading Hasse Diagrams

The Hasse diagram representation of a partial order

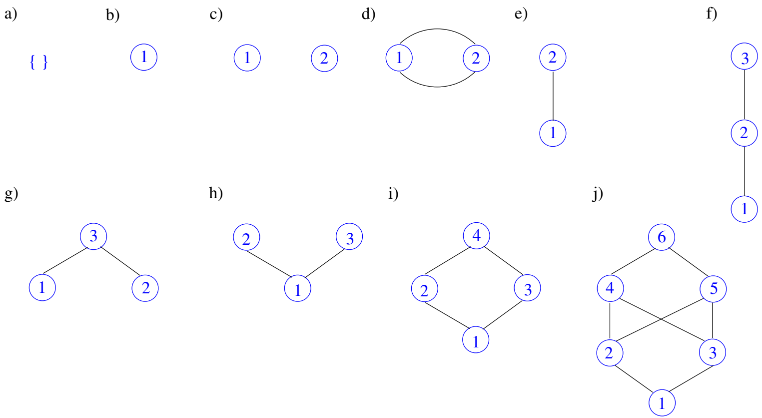
- omits links which express reflexive and transitive relations explicitly
- focuses on the 'immediate predecessor' relation.

The representation of a partial order by its Hasse diagram

- is sparse and thus economical (in the number of links).
- while preserving all relevant information of the partial order it represents:
  - $p \sqsubseteq q \wedge p = q$ (reflexivity): trivially represented (just without an explicit link)
  - $p \sqsubseteq q \wedge p \neq q$ (transitivity): represented by ascending paths (with at least one link) from p to q.

Exercise A.2.2.1

Which of the below diagrams are Hasse diagrams representing a partial order?



Exercise A.2.2.2

Let $|$ denote the divisibility relation on the set of natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \mid 35$.

Draw an expressive section of the Hasse diagram of the divisibility relation $|$ on $\mathbb{N}_0$.

A.2.3

Bounds and Extremal Elements

Bounds in Pre-Orders

Definition A.2.3.1 (Bounds in Pre-Orders)

Let $(Q, \sqsubseteq)$ be a pre-order, let $q \in Q$ and $Q' \subseteq Q$.

q is called a

- lower bound of Q', in signs: $q \sqsubseteq Q'$, if $\forall\, q' \in Q'.\ q \sqsubseteq q'$
- upper bound of Q', in signs: $Q' \sqsubseteq q$, if $\forall\, q' \in Q'.\ q' \sqsubseteq q$
- greatest lower bound (glb) (or infimum) of Q', in signs: $\bigsqcap Q'$, if q is a lower bound of Q' and for every other lower bound $\hat{q}$ of Q' holds: $\hat{q} \sqsubseteq q$.
- least upper bound (lub) (or supremum) of Q', in signs: $\bigsqcup Q'$, if q is an upper bound of Q' and for every other upper bound $\hat{q}$ of Q' holds: $q \sqsubseteq \hat{q}$.

Extremal Elements in Pre-Orders

Definition A.2.3.2 (Extremal Elements in Pre-Ord's)

Let $(Q, \sqsubseteq)$ be a pre-order, let $\sqsubset$ be the strict part of $\sqsubseteq$, and let $Q' \subseteq Q$ and $q \in Q'$.

q is called a

- minimal element of Q', if there is no $q' \in Q'$ with $q' \sqsubset q$.
- maximal element of Q', if there is no $q' \in Q'$ with $q \sqsubset q'$.
- least (or minimum) element of Q', if $q \sqsubseteq Q'$.
- greatest (or maximum) element of Q', if $Q' \sqsubseteq q$.

Note: Least and greatest elements of Q itself are usually denoted by $\perp$ and $\top$ (bottom, top (in German: Tief, Hoch)), respectively, if they exist. Least (greatest) elements of Q are always minimal (maximal) elements of Q.

Existence and Uniqueness

...of bounds and extremal elements in partially ordered sets.

Let $(P, \sqsubseteq)$ be a partial order, and let $Q \subseteq P$ be a subset of P.

Lemma A.2.3.3 (lub/glb: Unique if Existent)

Least upper bounds, greatest lower bounds, least elements, and greatest elements in Q are unique, if they exist.

Lemma A.2.3.4 (Minimal/Maximal El.: Not Unique)

Minimal and maximal elements in Q are usually not unique.

Note: Lemma A.2.3.3 suggests considering $\bigsqcup$ and $\bigsqcap$ partial maps $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ from the powerset $\mathcal{P}(P)$ of P to P.
Lemma A.2.3.3 does not hold for pre-orders.

Characterization of Least, Greatest Elements

...in terms of infima and suprema of sets.

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.3.5 (Characterization of $\perp$ and $\top$)

The least element $\perp$ and the greatest element $\top$ of P are given by the supremum and the infimum of the empty set, and the infimum and the supremum of P, respectively, i.e.,

$$\perp = \bigsqcup \emptyset = \bigsqcap P \ \text{ and } \ \top = \bigsqcap \emptyset = \bigsqcup P$$

if they exist.

Lower and Upper Bound Sets

Considering $\bigsqcup$ and $\bigsqcap$ partial functions $\bigsqcup, \bigsqcap : \mathcal{P}(P) \to P$ on the powerset of a partial order $(P, \sqsubseteq)$ suggests introducing two further maps $LB, UB : \mathcal{P}(P) \to \mathcal{P}(P)$ on $\mathcal{P}(P)$:

Definition A.2.3.6 (Lower and Upper Bound Sets)

Let $(P, \sqsubseteq)$ be a partial order. Then:

$LB, UB : \mathcal{P}(P) \to \mathcal{P}(P)$ denote two maps, which map a subset $Q \subseteq P$ to the set of its lower bounds and upper bounds, respectively:

1. $\forall\, Q \subseteq P.\ LB(Q) =_{df} \{lb \in P \mid \ lb \sqsubseteq Q\}$
2. $\forall\, Q \subseteq P.\ UB(Q) =_{df} \{ub \in P \mid Q \sqsubseteq ub\}$

Properties of Lower and Upper Bound Sets

Lemma A.2.3.7

Let $(P, \sqsubseteq)$ be a partial order, and let $Q \subseteq P$. Then:

$$\bigsqcup Q = \bigsqcap UB(Q) \quad \text{and} \quad \bigsqcap Q = \bigsqcup LB(Q)$$

if the supremum and the infimum of Q exist.

Lemma A.2.3.8

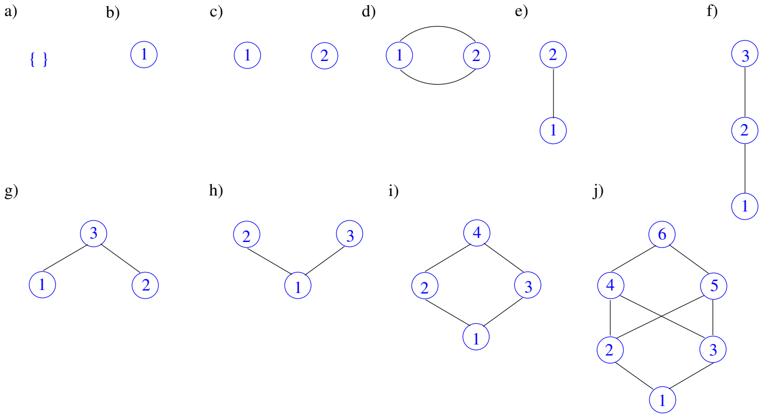
Let $(P, \sqsubseteq)$ be a partial order, and let $Q, Q_1, Q_2 \subseteq P$. Then:

1. $Q_1 \subseteq Q_2 \Rightarrow LB(Q_1) \supseteq LB(Q_2) \wedge UB(Q_1) \supseteq UB(Q_2)$
2. $UB(LB(UB(Q))) = UB(Q)$
3. $LB(UB(LB(Q))) = LB(Q)$

Note: Lemma A.2.3.8(1) shows that LB and UB are antitonic maps (cf. Chapter A.2.7).

Exercise A.2.3.9

Which of the elements of the below diagrams are minimal, maximal, least or greatest?



Exercise A.2.3.10

Let $|$ denote the divisibility relation on the set of natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \mid 35$.

Write down the sets of elements of $\mathbb{N}_0$, which are

1. minimal
2. maximal
3. least
4. greatest

wrt the divisibility relation $|$ on $\mathbb{N}_0$.

A.2.4

Noetherian and Artinian Orders

Noetherian and Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.4.1 (Noetherian Order)

$(P, \sqsubseteq)$ is called a Noetherian order, if every non-empty subset $\emptyset \neq Q \subseteq P$ contains a minimal element.

Definition A.2.4.2 (Artinian Order)

$(P, \sqsubseteq)$ is called an Artinian order, if the dual order $(P, \sqsupseteq)$ of $(P, \sqsubseteq)$ is a Noetherian order.

Lemma A.2.4.3

$(P, \sqsubseteq)$ is an Artinian order iff every non-empty subset $\emptyset \neq Q \subseteq P$ contains a maximal element.

Well-founded Orders

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.4.4 (Well-founded Order)

$(P, \sqsubseteq)$ is called a well-founded order, if $(P, \sqsubseteq)$ is a Noetherian order and totally ordered.

Lemma A.2.4.5

$(P, \sqsubseteq)$ is a well-founded order iff every non-empty subset $\emptyset \neq Q \subseteq P$ contains a least element.

Noetherian Induction

Theorem A.2.4.6 (Noetherian Induction)

Let $(N, \sqsubseteq)$ be a Noetherian order, let $N_{min} \subseteq N$ be the set of minimal elements of N, and let $\phi : N \to \mathbb{B}$ be a predicate on N. Then:

If

1. $\forall\, n \in N_{min}.\ \phi(n)$ (Induction base)
2. $\forall\, n \in N \backslash N_{min}.\ (\forall\, m \sqsubset n.\ \phi(m)) \Rightarrow \phi(n)$ (Induction step)

then:

$$\forall\, n \in N.\ \phi(n)$$

A.2.5

Chains

Chains, Antichains

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.5.1 (Chain)

A set $C \subseteq P$ is called a chain, if the elements of C are totally ordered, i.e., $\forall\, c_1, c_2 \in C.\ c_1 \sqsubseteq c_2 \vee c_2 \sqsubseteq c_1$.

Definition A.2.5.2 (Antichain)

A set $C \subseteq P$ is called an antichain, if
$\forall\, c_1, c_2 \in C.\ c_1 \sqsubseteq c_2 \Rightarrow c_1 = c_2$.

Definition A.2.5.3 (Finite, Infinite (Anti-) Chain)

Let $C \subseteq P$ be a chain or an antichain. C is called finite, if the number of its elements is finite; C is called infinite otherwise.

Note: Any set P may be converted into an antichain by giving it the discrete order: $(P, =)$.

Ascending Chains, Descending Chains

Definition A.2.5.4 (Ascending, Descending Chain)

Let $C \subseteq P$ be a chain. C given in the form of

- $C = \{c_0 \sqsubseteq c_1 \sqsubseteq c_2 \sqsubseteq \ldots\}$
- $C = \{c_0 \sqsupseteq c_1 \sqsupseteq c_2 \sqsupseteq \ldots\}$

is called an ascending chain and descending chain, respectively.

Examples of Chains

The set

- $S =_{df} \{n \in \mathbb{N} \mid n \ even\}$ is a chain in $\mathbb{N}$.
- $S =_{df} \{z \in \mathbb{Z} \mid z \ odd\}$ is a chain in $\mathbb{Z}$.
- $S =_{df} \{ \{k \in \mathbb{N} \mid k < n\} \mid n \in \mathbb{N}\}$ is a chain in the powerset $\mathcal{P}(\mathbb{N})$ of $\mathbb{N}$.

Note: A chain can always be given in the form of an ascending or descending chain.

- $\{0 \leq 2 \leq 4 \leq 6 \leq \ldots\}$: $\mathbb{N}$ as ascending chain.
- $\{\ldots \geq 6 \geq 4 \geq 2 \geq 0\}$: $\mathbb{N}$ as descending chain.
- $\{\ldots \leq -3 \leq -1 \leq 1 \leq 3 \leq \ldots\}$: $\mathbb{Z}$ as ascending chain.
- $\{\ldots \geq 3 \geq 1 \geq -1 \geq -3 \geq \ldots\}$: $\mathbb{Z}$ as descending chain.
- ...

Eventually Stationary Sequences

Definition A.2.5.5 (Stationary Sequence)

1. An ascending sequence of the form
$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots$$
is called eventually stationary, if
$$\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$$
2. A descending sequence of the form
$$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$
is called eventually stationary, if
$$\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$$

Chains and Sequences

Lemma A.2.5.6

An ascending or descending sequence of the form

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{or} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

1. is a finite chain iff it is eventually stationary.
2. is an infinite chain iff it is not eventually stationary.

Note the subtle difference between the notion of chains in terms of sets

$$\{p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots\} \quad \text{or} \quad \{p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots\}$$

and in terms of sequences

$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots \quad \text{or} \quad p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$

Sequences may contain duplicates, which would correspond to defining chains in terms of multisets.

Ascending, Descending Chain Condition

Let $(P, \sqsubseteq)$ be a partial order.

Definition A.2.5.7 (Asc./Desc. Chain Condition)

$(P, \sqsubseteq)$ satisfies the

1. ascending chain condition (in German: aufsteigende Kettenbedingung), if every ascending chain is eventually stationary, i.e., for every chain $p_1 \sqsubseteq p_2 \sqsubseteq \ldots \sqsubseteq p_n \sqsubseteq \ldots$ there is an index $m \geq 1$ with $p_m = p_{m+j}$ for all $j \in \mathbb{N}$.
2. descending chain condition (in German: absteigende Kettenbedingung), if every descending chain is eventually stationary, i.e., for every chain $p_1 \sqsupseteq p_2 \sqsupseteq \ldots \sqsupseteq p_n \sqsupseteq \ldots$ there is an index $m \geq 1$ with $p_m = p_{m+j}$ for all $j \in \mathbb{N}$.

Chains and Noetherian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.5.8 (Noetherian Order)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is a Noetherian order.
2. $(P, \sqsubseteq)$ satisfies the descending chain condition.
3. Every chain of the form
$$p_0 \sqsupseteq p_1 \sqsupseteq p_2 \sqsupseteq \ldots$$
is eventually stationary, i.e.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
4. Every chain of the form
$$p_0 \sqsupset p_1 \sqsupset p_2 \sqsupset \ldots$$
is finite.

Chains and Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.5.9 (Artinian Order)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is an Artinian order.
2. $(P, \sqsubseteq)$ satisfies the ascending chain condition.
3. Every chain of the form
$$p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots$$
is eventually stationary, i.e.: $\exists\, n \in \mathbb{N}.\ \forall\, j \in \mathbb{N}.\ p_{n+j} = p_n$.
4. Every chain of the form
$$p_0 \sqsubset p_1 \sqsubset p_2 \sqsubset \ldots$$
is finite.

Chains and Noetherian, Artinian Orders

Let $(P, \sqsubseteq)$ be a partial order.

Lemma A.2.5.10 (Noetherian and Artinian Order)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is a Noetherian and an Artinian order.
2. $(P, \sqsubseteq)$ satisfies the descending and the ascending chain condition.
3. Every chain $C \subseteq P$ is finite.

A.2.6

Directed Sets

Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $\emptyset \neq D \subseteq P$.

Definition A.2.6.1 (Directed Set)

D $(\neq \emptyset)$ is called a directed set (in German: gerichtete Menge), if

$$\forall\, d, e \in D.\ \exists\, f \in D.\ f \in UB(\{d, e\})$$

i.e., for any two elements d and e there is a common upper bound of d and e in D, i.e., $UB(\{d, e\}) \cap D \neq \emptyset$.

Properties of Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Lemma A.2.6.2

D is a directed set iff any finite subset $D' \subseteq D$ has an upper bound in D, i.e., $\exists\, d \in D.\ d \in UB(D')$, i.e., $UB(D') \cap D \neq \emptyset$.

Lemma A.2.6.3

If D has a greatest element, then D is a directed set.

Properties of Finite Directed Sets

Let $(P, \sqsubseteq)$ be a partial order, and let $D \subseteq P$.

Corollary A.2.6.4

Let D be a finite directed set. Then: $\bigsqcup D$ *exists* $\in D$ and is the greatest element of D.

Proof. Since D a directed set, we have:

$$\exists\, d \in D.\ d \in UB(D), \text{ i.e., } UB(D) \cap D \neq \emptyset.$$

This means $D \sqsubseteq d$. The antisymmetry of $\sqsubseteq$ yields that the element enjoying this property is unique. Thus, d is the (unique) greatest element of D given by $\bigsqcup D$, i.e., $d = \bigsqcup D$.

Note: If D is infinite, the statement of Corollary A.2.6.4 does usually not hold.

Strongly Directed Sets

Let $(P, \sqsubseteq)$ be a partial order with least element $\perp$, and let $D \subseteq P$.

Definition A.2.6.5 (Strongly Directed Set)

$D \neq \emptyset$ is called a strongly directed set (in German: stark gerichtete Menge), if

1. $\perp \in D$
2. $\forall\, d, e \in D.\ \exists\, f \in D.\ f = \bigsqcup\{d, e\}$, i.e., for any two elements d and e the supremum $\bigsqcup\{d, e\}$ of d and e exists in D.

Properties of Strongly Directed Sets

Let $(P, \sqsubseteq)$ be a partial order with least element $\perp$, and let $D \subseteq P$.

Lemma A.2.6.6

D is a strongly directed set iff every finite subset $D' \subseteq D$ has a supremum in D, i.e., $\exists\, d \in D.\ d = \bigsqcup D'$.

Lemma A.2.6.7

Let D be a finite strongly directed set. Then: $\bigsqcup D$ *exists* $\in D$ and is the greatest element of D.

Note: The statement of Lemma A.2.6.7 does usually not hold, if D is infinite.

Directed Sets, Strongly Directed Sets, Chains

Let $(P, \sqsubseteq)$ be a partial order with least element $\perp$.

Lemma A.2.6.8

Let $\emptyset \neq D \subseteq P$ be a non-empty subset of P. Then:

1. D is a directed set, if D is a strongly directed set.
2. D is a strongly directed set, if $\perp \in D$ and D is a chain.

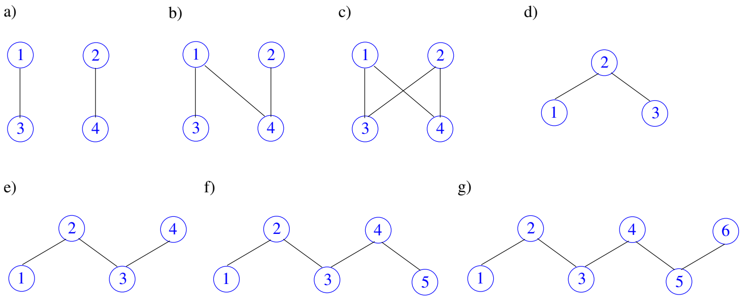
Corollary A.2.6.9

Let $\emptyset \neq D \subseteq P$ be a non-empty subset of P. Then:

$\perp \in D \wedge D$ chain $\Rightarrow$ D strongly directed set $\Rightarrow$ D directed set

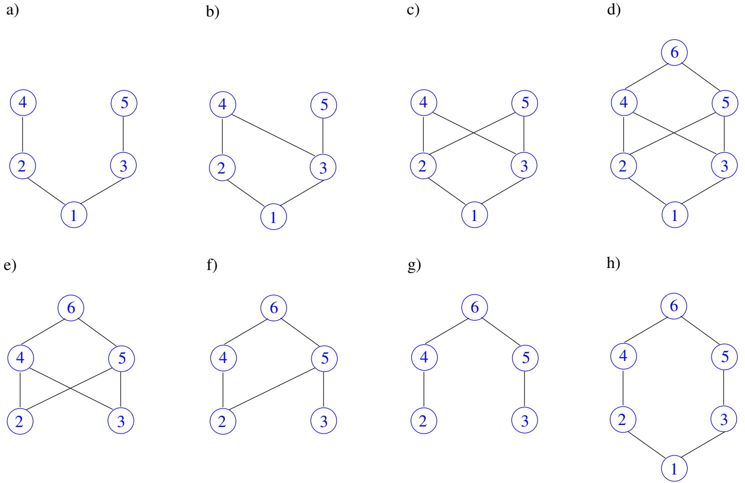
Exercise A.2.6.10

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?



Exercise A.2.6.11

Which of the below partial orders are (strongly) directed sets?
Which of their subsets are (strongly) directed sets?



Exercise A.2.6.12

Let $(\mathrm{IN}_0, \sqsubseteq)$ be the partial order with $\sqsubseteq =_{df} |$, where $|$ denotes the divisibility relation on the natural numbers IN_0, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \,|\, 35$.

Is the set IN_0

1. directed?
2. strongly directed?

What subsets of IN_0 are

1. directed?
2. strongly directed?

Proof or counterexample.

A.2.7

Maps on Partial Orders

Monotonic and Antitonic Maps on POs

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be partial orders, and let $f \in [C \to D]$ be a map from C to D.

Definition A.2.7.1 (Monotonic Maps on POs)

f is called monotonic (or order preserving) iff

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$$

(Preservation of the ordering of elements)

Definition A.2.7.2 (Antitonic Maps on POs)

f is called antitonic (or order inversing) iff

$$\forall\, c, c' \in C.\ c \sqsubseteq_C c' \Rightarrow f(c') \sqsupseteq_D f(c)$$

(Inversion of the ordering of elements)

Expanding and Contracting Maps on POs

Let $(C, \sqsubseteq_C)$ be a partial order, let $f \in [C \to C]$ be a map on C, and let $\hat{c} \in C$ be an element of C.

Definition A.2.7.3 (Expanding Maps on POs)

f is called

- expanding (or inflationary) for $\hat{c}$ iff $\hat{c} \sqsubseteq f(\hat{c})$
- expanding (or inflationary) iff $\forall\, c \in C.\ c \sqsubseteq f(c)$

Definition A.2.7.4 (Contracting Maps on POs)

f is called

- contracting (or deflationary) for $\hat{c}$ iff $f(\hat{c}) \sqsubseteq \hat{c}$
- contracting (or deflationary) iff $\forall\, c \in C.\ f(c) \sqsubseteq c$

A.2.8

Order Homomorphisms and Order Isormorphisms

PO Homomorphisms, PO Isomorphisms

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be partial orders, and let $f \in [P \to R]$ be a map from P to R.

Definition A.2.8.1 (PO Hom. & Isomorphism)

f is called an

1. order homomorphism between P and R, if f is monotonic (or order preserving), i.e.,
$$\forall\, p, q \in P.\ p \sqsubseteq_P q \Rightarrow f(p) \sqsubseteq_R f(q)$$
2. order isomorphism between P and R, if f is a bijective order homomorphism between P and R and the inverse f^{-1} of f is an order homomorphism between R and P.

Definition A.2.8.2 (Order Isomorphic)

$(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ are called order isomorphic, if there is an order isomorphism between P and R.

PO Embeddings

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be partial orders, and let $f \in [P \to R]$ be a map from P to R.

Definition A.2.8.3 (PO Embedding)

f is called an order embedding of P in R iff

$$\forall\, p, q \in P.\ p \sqsubseteq_P q \iff f(p) \sqsubseteq_R f(q)$$

Lemma A.2.8.4 (PO Embeddings and Isomorphisms)

f is an order isomorphism between P and R iff f is an order embedding of P in R and f is surjective.

Intuitively: Partial orders, which are order isomorphic, are 'essentially the same.'

A.3

Complete Partially Ordered Sets

A.3.1

Chain and Directly Complete Partial Orders

Complete Partially Ordered Sets

...or Complete Partial Orders:

- a slightly weaker ordering notion than that of a lattice (cf. Appendix A.4), which is often more adequate for the modelling of problems in computer science, where full lattice properties are often not required.
- come in two different flavours as so-called
  - Chain Complete Partial Orders (CCPOs)
  - Directedly Complete Partial Orders (DCPOs)

  based on the notions of chains and directed sets, respectively, which, however, are equivalent (cf. Theorem 3.1.7).

Complete Partial Orders: CCPO View

Definition A.3.1.1 (Chain Complete Partial Order)

A partial order $(P, \sqsubseteq)$ is a

1. chain complete partial order (pre-CCPO), if every non-empty (ascending) chain $\emptyset \neq C \subseteq P$ has a least upper bound $\bigsqcup C$ in P, i.e., $\bigsqcup C$ *exists* $\in P$.
2. pointed chain complete partial order (CCPO), if every (ascending) chain $C \subseteq P$ has a least upper bound $\bigsqcup C$ in P, i.e., $\bigsqcup C$ *exists* $\in P$.

Note: Some authors use CCPO and CCPPO instead of pre-CCPO and CCPO, respectively.

Complete Partial Orders: DCPO View

Definition A.3.1.2 (Directedly Complete Partial Ord.)

A partial order $(P, \sqsubseteq)$ is a

1. directedly complete partial order (pre-DCPO), if every directed subset $D \subseteq P$ has a least upper bound $\bigsqcup D$ in P, i.e., $\bigsqcup D$ *exists* $\in P$.
2. pointed directedly complete partial order (DCPO), if it is a pre-DCPO and has a least element $\perp$.

Note: Some authors use DCPO and DCPPO instead of pre-DCPO and DCPO, respectively.

Remarks on CCPOs and DCPOs

On CCPOs:

- A CCPO is often called a domain.
- 'Ascending chain' and 'chain' can equivalently be used in Definition A.3.1.1, since a chain can always be given in ascending order. 'Ascending' chain is just more intuitive.

On DCPOs:

- A directed set S, in which by definition every finite subset has an upper bound in S, does not need to have a supremum in S, if S is infinite. Therefore, the DCPO property does not trivially follow from the directed set property (cf. Corollary A.2.6.4).

Existence of Least Elements in CPOs

Lemma A.3.1.3 (Least Elem. Existence in CPOs)

Let $(C, \sqsubseteq)$ be a CPO, i.e., a CCPO or DCPO. Then there is a unique least element in C, denoted by $\perp$, which is given by the supremum of the empty chain or set, i.e.: $\perp = \bigsqcup \emptyset$.

Corollary A.3.1.4 (Non-Emptyness of CPOs)

Let $(C, \sqsubseteq)$ be a CPO, i.e., a CCPO or DCPO. Then: $C \neq \emptyset$.

Note: Lemma A.3.1.3 does not hold for pre-CPOs, i.e., a pre-CPO $(P, \sqsubseteq)$ does not need to have a least element.

Relating Finite POs, CCPOs and DCPOs

Let P be a finite set, and let $\sqsubseteq$ be a relation on P.

Lemma A.3.1.5 (Fin. POs, pre-CCPOs, pre-DCPOs)

The following statements are equivalent:

1. $(P, \sqsubseteq)$ is a partial order.
2. $(P, \sqsubseteq)$ is a pre-CCPO.
3. $(P, \sqsubseteq)$ is a pre-DCPO.

Lemma A.3.1.6 (Finite POs, CCPOs, DCPOs)

Let $p \in P$ with $p \sqsubseteq P$. Then the following statements are equivalent:

1. $(P, \sqsubseteq)$ is a partial order.
2. $(P, \sqsubseteq)$ is a CCPO.
3. $(P, \sqsubseteq)$ is a DCPO.

Equivalence of CCPOs and DCPOs

Theorem A.3.1.7 (Equivalence)

Let $(P, \sqsubseteq)$ be a partial order. Then the following statements are equivalent:

1. $(P, \sqsubseteq)$ is a CCPO.
2. $(P, \sqsubseteq)$ is a DCPO.

Note: We simply speak of a CPO, if its flavour based on chains (CCPO) or directed sets (DCPO) does not matter; analogously, this applies to pre-CPOs.

Examples of pre-CPOs and CPOs (1)

- $(\mathcal{P}(\mathbb{N}), \subseteq)$ is a CPO (i.e., a CCPO and a DCPO).
  - Least element: $\emptyset$
  - Least upper bound $\bigsqcup C$ of C *chain* $\subseteq \mathcal{P}(\mathbb{N})$: $\bigcup_{C' \in C} C'$
- The set of finite and infinite strings S partially ordered by the prefix relation $\sqsubseteq_{pfx}$ defined by

$$\forall\, s, s'' \in S.\ s \sqsubseteq_{pfx} s'' \iff_{df} s = s'' \vee (s \textit{ finite } \wedge \exists\, s' \in S.\ s +\!\!+ s' = s'')$$

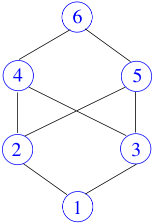
  is a CPO.
- $(\{-n \mid n \in \mathbb{N}\}, \leq)$ is a pre-CPO (i.e., a pre-CCPO and a pre-DCPO) but not a CPO (i.e., not a CCPO and DCPO).

Examples of pre-CPOs and CPOs (2)

- $(\emptyset, \emptyset)$ is a pre-CPO (i.e., a pre-CCPO and a pre-DCPO) but not a CPO (i.e., not a CCPO and DCPO).

  (Both the pre-CCPO (absence of non-empty chains in $\emptyset$) and the pre-DCPO ($\emptyset$ is the only subset of $\emptyset$ and is not directed by definition) property holds trivially. Note also that $P = \emptyset$ implies $\sqsubseteq = \emptyset \subseteq P \times P$).

- The partial order $(P \sqsubseteq)$ given by the below Hasse diagram is a CPO.



Examples of pre-CPOs and CPOs (3)

- The set of finite and infinite strings S partially ordered by the lexicographical order $\sqsubseteq_{lex}$ defined by

  $$\forall\, s,t \in S.\ s \sqsubseteq_{lex} t \iff_{df}$$
  $$s = t \vee (\exists\, p\ finite, s', t' \in S.\ s = p +\!\!+ s' \ \wedge\ t = p +\!\!+ t' \ \wedge$$
  $$(s' = \varepsilon \ \vee\ s'_1 < t'_1)\,)$$

  where ε denotes the empty string, $w\!\downarrow_1$ denotes the first character of a string w, and $<$ the lexicographical ordering on characters, is a CPO (i.e., a CCPO and a DCPO).

(Anti-) Examples of CPOs

- $(\mathbb{N}, \leq)$ is not a CPO (i.e., not a CCPO and DCPO).
- The set of finite strings S_{fin} partially ordered by the
  - prefix relation $\sqsubseteq_{pfx}$ defined by

    $$\forall\, s, s' \in S_{fin}.\ s \sqsubseteq_{pfx} s' \iff_{df} \exists\, s'' \in S_{fin}.\ s +\!\!+ s'' = s'$$

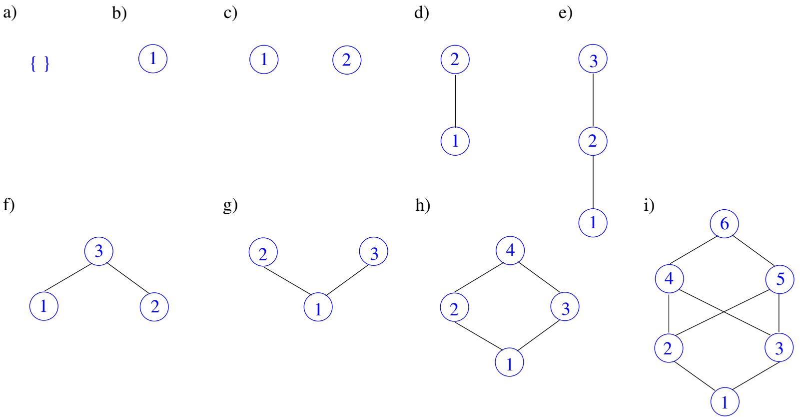
    is not a CPO (i.e., not a CCPO and DCPO).
  - lexicographical order $\sqsubseteq_{lex}$ defined by

    $$\begin{aligned}\forall\, s, t \in S_{fin}.\ s \sqsubseteq_{lex} t \iff_{df} & \\ \exists\, p, s', t' \in S_{fin}.\ & s = p +\!\!+ s' \ \wedge\ t = p +\!\!+ t' \ \wedge \\ & (s' = \varepsilon \ \vee\ s'\!\downarrow_1 \ < t'\!\downarrow_1)\end{aligned}$$

    where ε denotes the empty string, $w\!\downarrow_1$ denotes the first character of a string w, and $<$ the lexicographical ordering on characters, is not a CPO (i.e., not a CCPO and DCPO).
- $(\mathcal{P}_{fin}(\mathbb{N}), \subseteq)$ is not a CPO (i.e., not a CCPO and DCPO).

Exercise A.3.1.8

Which of the partial orders given by the below Hasse diagrams are (pre-) CCPOs? Which ones are (pre-) DCPOs?



Strongly Directed CPOs: A DCPO Variant

On DCPOs based on Strongly Directed Sets

- Replacing directed sets by strongly directed sets in Definition A.3.1.2 leads to SDCPOs.
- Recalling that strongly directed sets are not empty (cf. Definition A.2.6.5), there is no analogue of pre-DCPOs for strongly directed sets.
- A strongly directed set S, in which by definition every finite subset has a supremum in S, does not need to have a supremum itself in S, if S is infinite. Therefore, the SDCPO property does not trivially follow from the strongly directed property of sets (cf. Corollary A.2.6.4).

Exercise A.3.1.9

Let $(\mathbb{N}_0, \sqsubseteq)$ be the partial order with $\sqsubseteq =_{df} |$, where $|$ denotes the divisibility relation on the natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \,|\, 35$.

Prove or disprove: $(\mathbb{N}_0, \sqsubseteq)$ is a

1. pre-CCPO
2. CCPO
3. pre-DCPO
4. DCPO
5. SDCPO

Proof or counterexample.

A.3.2

Maps on Complete Partial Orders

Continuous Maps on CCPOs

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be CCPOs, and let $f \in [C \to D]$ be a map from C to D.

Definition A.3.2.1 (Continuous Maps on CCPOs)

f is called continuous iff f is monotonic and

$$\forall\, C' \neq \emptyset \; chain \subseteq C.\; f(\bigsqcup_C C') \;=_D\; \bigsqcup_D f(C')$$

(Preservation of least upper bounds)

Note: $\forall\, S \subseteq C.\; f(S) =_{df} \{\, f(s) \,|\, s \in S \}$

Continuous Maps on DCPOs

Let $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$ be DCPOs, and let $f \in [D \to E]$ be a map from D to E.

Definition A.3.2.2 (Continuous Maps on DCPOs)

f is called continuous iff

$$\forall\, D' \neq \emptyset \textit{ directed set} \subseteq D.\ f(D') \text{ directed set } \subseteq E \wedge$$
$$f(\textstyle\bigsqcup_D D') =_E \bigsqcup_E f(D')$$

(Preservation of least upper bounds)

Note: $\forall\, S \subseteq D.\ f(S) =_{df} \{\, f(s) \mid s \in S \}$

Characterizing Monotonicity

Let $(C, \sqsubseteq_C), (D, \sqsubseteq_D)$ be CCPOs, let $(E, \sqsubseteq_E), (F, \sqsubseteq_F)$ be DCPOs.

Lemma A.3.2.3 (Characterizing Monotonicity)

1. $f : C \to D$ is monotonic
   iff $\forall\, C' \neq \emptyset$ *chain* $\subseteq C.$
   $$f(C') \text{ chain} \subseteq D \wedge f(\bigsqcup_C C') \sqsupseteq_D \bigsqcup_D f(C')$$
2. $g : E \to F$ is monotonic
   iff $\forall\, E' \neq \emptyset$ *directed set* $\subseteq E.$
   $$g(E') \text{ directed set} \subseteq F \wedge g(\bigsqcup_E E') \sqsupseteq_F \bigsqcup_F g(E')$$

Strict Maps on CCPOs and DCPOs

Let $(C, \sqsubseteq_C), (D, \sqsubseteq_D)$ be CCPOs with least elements $\perp_C$ and $\perp_D$, respectively, let $(E, \sqsubseteq_E), (F, \sqsubseteq_F)$ be DCPOs with least elements $\perp_E$ and $\perp_F$, respectively, and let $f \in [C \stackrel{con}{\to} D]$ and $g \in [E \stackrel{con}{\to} F]$ be continuous maps.

Definition A.3.2.4 (Strict Functions on CPOs)

f and g are called strict, if the equalities

- $f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$, $g(\bigsqcup_E E') =_F \bigsqcup_F g(E')$

also hold for $C' = \emptyset$ and $E' = \emptyset$, i.e., if the equalities

- $f(\bigsqcup_C \emptyset) =_C \; f(\perp_C) =_D \perp_D \; =_D \bigsqcup \emptyset$
- $f(\bigsqcup_E \emptyset) =_E \; g(\perp_E) =_F \perp_F \; =_F \bigsqcup \emptyset$

are valid.

A.3.3

Mechanisms for Constructing Complete Partial Orders

Common CCPO and DCPO Constructions

The following construction principles hold for

- CCPOs
- DCPOs

Therefore, we simply write CPO.

Common CPO Constructions: Flat CPOs

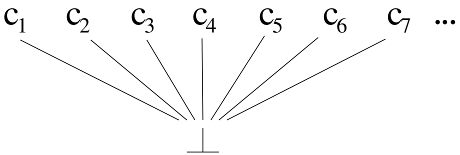
Lemma A.3.3.1 (Flat CPO Construction)

Let C be a set. Then:

$(C \mathbin{\dot\cup} \{\bot\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, c, d \in C \mathbin{\dot\cup} \{\bot\}.\; c \sqsubseteq_{flat} d \iff_{df} c = \bot \;\vee\; c = d$$

is a CPO, a so-called flat CPO.



Common CPO Constructions: Flat pre-CPOs

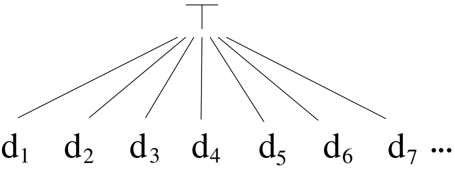
Lemma A.3.3.2 (Flat Pre-CPO Construction)

Let D be a set. Then:

$(D \,\dot\cup\, \{\top\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, d, e \in D \,\dot\cup\, \{\top\}.\ d \sqsubseteq_{flat} e \iff_{df} e = \top \ \vee\ d = e$$

is a pre-CPO, a so-called flat pre-CPO.



Contents
Part III
Chap. 7
Chap. 10
Appendices
A
A.1
A.2
A.3
A.3.1
A.3.2
A.3.3
A.4
A.5
A.6
A.7
B

Common CPO Constructions: Products (1)

Lemma A.3.3.3 (Non-strict Product Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The non-strict product $(\times P_i, \sqsubseteq_\times)$, where

- $\times P_i =_{df} P_1 \times P_2 \times \ldots \times P_n$ is the cartesian product of all P_i, $1 \leq i \leq n$
- $\sqsubseteq_\times$ is defined pointwise by

$$\forall\, (p_1, \ldots, p_n),\, (q_1, \ldots, q_n) \in \times P_i.$$
$$(p_1, \ldots, p_n) \sqsubseteq_\times (q_1, \ldots, q_n) \iff_{df} \forall\, i \in \{1, \ldots, n\}.\ p_i \sqsubseteq_i q_i$$

is a CPO.

Common CPO Constructions: Products (2)

Lemma A.3.3.4 (Strict Product Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The strict (or smash) product $(\bigotimes P_i, \sqsubseteq_\otimes)$, where

- $\bigotimes P_i =_{df} \times P_i$ is the the cartesian product of all P_i
- $\sqsubseteq_\otimes =_{df} \sqsubseteq_\times$ defined pointwise with the additional setting

$$(p_1, \ldots, p_n) = \perp \iff_{df} \exists\, i \in \{1, \ldots, n\}.\ p_i = \perp_i$$

is a CPO.

Common CPO Constructions: Sums (1)

Lemma A.3.3.5 (Separated Sum Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The separated (or direct) sum $(\bigoplus_\perp P_i, \sqsubseteq_{\oplus_\perp})$, where

- $\bigoplus_\perp P_i =_{df} P_1 \dot\cup P_2 \dot\cup \ldots \dot\cup P_n \dot\cup \{\perp\}$ is the disjoint union of all P_i, $1 \leq i \leq n$, and a fresh bottom element $\perp$
- $\sqsubseteq_{\oplus_\perp}$ is defined by
  $\forall\, p, q \in \bigoplus_\perp P_i.\ p \sqsubseteq_{\oplus_\perp} q \iff_{df}$
  $$p = \perp\ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i\ \wedge\ p \sqsubseteq_i q)$$

is a CPO.

Common CPO Constructions: Sums (2)

Lemma A.3.3.6 (Coalesced Sum Construction)

Let $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \ldots, (P_n, \sqsubseteq_n)$ be CPOs. Then:

The coalesced sum $(\bigoplus_\vee P_i, \sqsubseteq_{\oplus\vee})$, where

- $\bigoplus_\vee P_i =_{df} P_1 \backslash \{\perp_1\} \dot\cup P_2 \backslash \{\perp_2\} \,\dot\cup \ldots \dot\cup\, P_n \backslash \{\perp_n\} \,\dot\cup\, \{\perp\}$ is the disjoint union of all P_i, $1 \leq i \leq n$, and a fresh bottom element $\perp$, which is identified with and replaces the least elements $\perp_i$ of the sets P_i, i.e., $\perp =_{df} \perp_i$, $i \in \{1, \ldots, n\}$
- $\sqsubseteq_{\oplus\vee}$ is defined by
$$\forall\, p, q \in \bigoplus_\vee P_i.\ p \sqsubseteq_{\oplus\vee} q \iff_{df}$$
$$p = \perp \ \vee\ (\exists\, i \in \{1, \ldots, n\}.\ p, q \in P_i \ \wedge\ p \sqsubseteq_i q)$$

is a CPO.

Common CPO Constructions: Function Space

Lemma A.3.3.7 (Continuous Function Space Con.)

Let $(C, \sqsubseteq_C)$ and $(D, \sqsubseteq_D)$ be pre-CPOs. Then:

The continuous function space $([C \stackrel{con}{\to} D], \sqsubseteq_{cfs})$, where

- $[C \stackrel{con}{\to} D]$ is the set of continuous maps from C to D
- $\sqsubseteq_{cfs}$ is defined pointwise by
  $\forall\, f, g \in [C \stackrel{con}{\to} D].\ f \sqsubseteq_{cfs} g \iff_{df} \forall\, c \in C.\ f(c) \sqsubseteq_D g(c)$

is a pre-CPO. It is a CPO, if $(D, \sqsubseteq_D)$ is a CPO.

Note: The definition of $\sqsubseteq_{cfs}$ does not make use of C being a pre-CPO. This requirement is only to allow us tailoring the definition to continuous maps.

A.4

Lattices

A.4.1

Lattices, Complete Lattices

Lattices and Complete Lattices

Let $(P, \sqsubseteq)$ be a partial order, $P \neq \emptyset$.

Definition A.4.1.1 (Lattice)

$(P, \sqsubseteq)$ is a lattice (in German: Verband), if every non-empty finite subset P' of P has a least upper bound and a greatest lower bound in P.

Definition A.4.1.2 (Complete Lattice)

$(P, \sqsubseteq)$ is a complete lattice (in German: vollständiger Verband), if every subset P' of P has a least upper bound and a greatest lower bound in P.

Note: Lattices and complete lattices are special partial orders.

Properties of Complete Lattices

Lemma A.4.1.3 (Existence of Extremal Elements)

Let $(P, \sqsubseteq)$ be a complete lattice. Then there is

1. a least element in P, denoted by $\bot$, satisfying: $\bot = \bigsqcup \emptyset = \bigsqcap P$.
2. a greatest element in P, denoted by $\top$, satisfying: $\top = \bigsqcap \emptyset = \bigsqcup P$.

Lemma A.4.1.4 (Characterization Lemma)

Let $(P, \sqsubseteq)$ be a partial order. Then the following statements are equivalent:

1. $(P, \sqsubseteq)$ is a complete lattice.
2. Every subset of P has a least upper bound.
3. Every subset of P has a greatest lower bound.

Properties of Finite Lattices

Lemma A.4.1.5 (Finiteness implies Completeness)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ is a complete lattice.

Corollary A.4.1.6 (Finiteness impl. Ex. of ext. Elem.)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ has a least element and a greatest element.

Complete Semi-Lattices

Let $(P, \sqsubseteq)$ be a partial order, $P \neq \emptyset$.

Definition A.4.1.7 (Complete Semi-Lattice)

$(P, \sqsubseteq)$ is a complete

1. join semi-lattice (in German: Vereinigungshalbverband) iff $\forall \emptyset \neq S \subseteq P.\ \bigsqcup S \ \textit{exists} \in P$.
2. meet semi-lattice (in German: Schnitthalbverband) iff $\forall \emptyset \neq S \subseteq P.\ \bigsqcap S \ \textit{exists} \in P$.

Properties of Complete Semi-Lattices (1)

Proposition A.4.1.8 (Extr. Bounds in C. Semi-Lat.)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice, then $\bigsqcup P$ *exists* $\in P$ (whereas $\bigsqcup \emptyset$ ($\widehat{=} \bot$) does usually not exist in P).
2. meet semi-lattice, then $\bigsqcap P$ *exists* $\in P$ (whereas $\bigsqcap \emptyset$ ($\widehat{=} \top$) does usually not exist in P).

Informally: Least elements need not exist in complete join semi-lattices, greatest elements need not exist in complete meet semi-lattices.

Properties of Complete Semi-Lattices (2)

Lemma A.4.1.9 (Ex. great. El. in C. Join Semi-Lat.)

Let $(P, \sqsubseteq)$ be a complete join semi-lattice. Then:

$\bigsqcup P$ *exists* $\in P$ and is the (unique) greatest element in P that is usually denoted by $\top$, i.e., $\top = \bigsqcup P$.

Lemma A.4.1.10 (Ex. least El. in C. Meet Semi-Lat.)

Let $(P, \sqsubseteq)$ be a complete meet semi-lattice. Then:

$\bigsqcap P$ *exists* $\in P$ and is the (unique) least element in P that is usually denoted by $\bot$, i.e., $\bot = \bigsqcap P$.

Characterizing Upper and Lower Bounds (1)

...in complete semi-lattices.

Lemma A.4.1.11 (Char. u./l. Bounds in C. Semi-L.)

1. Let $(P, \sqsubseteq)$ be a complete join semi-lattice, and let $Q \subseteq P$ be a subset of P.

   If there is a lower bound for Q in P, i.e, if $\{p \in P \mid p \sqsubseteq Q\} \neq \emptyset$, then $\bigsqcap Q$ *exists* $\in P$ satisfying

   $$\bigsqcap Q = \bigsqcup \{p \in P \mid p \sqsubseteq Q\}$$

2. Let $(P, \sqsubseteq)$ be a complete meet semi-lattice, and let $Q \subseteq P$ be a subset of P.

   If there is an upper bound for Q in P, i.e, if $\{p \in P \mid Q \sqsubseteq p\} \neq \emptyset$, then $\bigsqcup Q$ *exists* $\in P$ satisfying

   $$\bigsqcup Q = \bigsqcap \{p \in P \mid Q \sqsubseteq p\}$$

Characterizing Upper and Lower Bounds (2)

Lemma A.4.1.12 (L./gr. Elements in C. Semi-L.)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice and $\bigsqcup \emptyset$ *exists* $\in P$, then $\bigsqcup \emptyset$ is the (unique) least element in P, denoted by $\perp$, i.e., $\perp = \bigsqcup \emptyset$.
2. meet semi-lattice and $\bigsqcap \emptyset$ *exists* $\in P$, then $\bigsqcap \emptyset$ is the (unique) greatest element in P, denoted by $\top$, i.e., $\top = \bigsqcap \emptyset$.

Relating Complete Semi-Lattices and Lattices

Lemma A.4.1.13 (Complete Semi-Lattices&Lattices)

If $(P, \sqsubseteq)$ is a complete

1. join semi-lattice and $\bigsqcup \emptyset$ *exists* $\in P$
2. meet semi-lattice and $\bigsqcap \emptyset$ *exists* $\in P$

then $(P, \sqsubseteq)$ is a complete lattice.

Exercise A.4.1.14

Prove or disprove:

If $(P, \sqsubseteq)$ is a complete lattice, then

1. $(P \backslash \{\bot\}, \sqsubseteq_{\backslash\bot})$ is a complete join semi-lattice.
2. $(P \backslash \{\top\}, \sqsubseteq_{\backslash\top})$ is a complete meet semi-lattice.

where $\sqsubseteq_{\backslash\bot}$ and $\sqsubseteq_{\backslash\top}$ denote the restrictions of $\sqsubseteq$ from P to $P \backslash \{\bot\}$ and $P \backslash \{\top\}$, respectively. Proof or counterexample.

Relating Lattices and Complete Partial Orders

Lemma A.4.1.15 (Complete Lattices and CPOs)

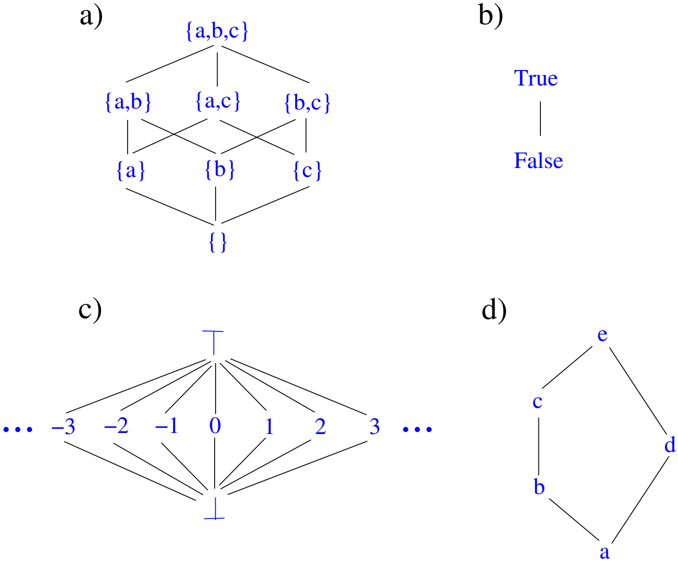
If $(P, \sqsubseteq)$ is a complete lattice, then $(P, \sqsubseteq)$ is a CPO (i.e., a CCPO and DCPO).

Corollary A.4.1.16 (Finite Lattices and CPOs)

If $(P, \sqsubseteq)$ is a finite lattice, then $(P, \sqsubseteq)$ is a CPO (i.e., a CCPO and DCPO).

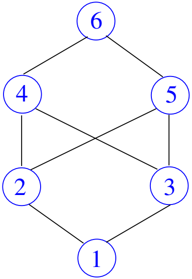
Note: Lemma A.4.1.15 does not hold for lattices.

Examples of Complete Lattices



(Anti-) Examples

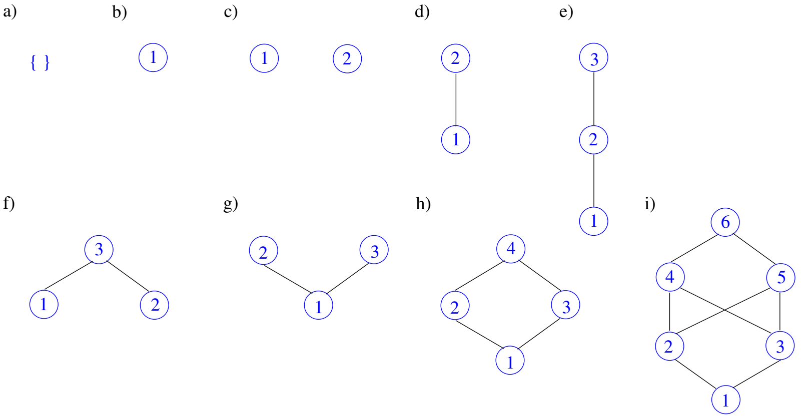
- The partial order $(P, \sqsubseteq)$ given by the below Hasse diagram is not a lattice (whereas it is a CPO).



- $(\mathcal{P}_{fin}(\mathbb{N}), \subseteq)$ is not a complete lattice (and not a CPO).

Exercise A.4.1.17

Which of the partial orders given by the below Hasse diagrams are lattices? Which ones are complete lattices?



Exercise A.4.1.18

Let $(\mathbb{N}_0, \sqsubseteq)$ be the partial order with $\sqsubseteq =_{df} |$, where $|$ denotes the divisibility relation on the natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \,|\, 35$.

Prove or disprove: $(\mathbb{N}_0, \sqsubseteq)$ is a

1. lattice
2. complete lattice
3. complete join semi-lattice
4. complete meet semi-lattice

Proof or counterexample.

Summary, Overview

Corollary A.4.1.19

Let $P \neq \emptyset$ be a non-empty set, and $\sqsubseteq$ a relation on P. Then:

$$
\begin{array}{rcl}
 & & (P, \sqsubseteq) \text{ finite lattice (L. A.4.1.5)} \vee \\
 & & (P, \sqsubseteq) \text{ complete join semi-lattice and} \\
 & & \quad \bigsqcup \emptyset \textit{ exists} \in P \text{ (L. A.4.1.13(1))} \vee \\
 & & (P, \sqsubseteq) \text{ complete meet semi-lattice and} \\
 & & \quad \bigsqcap \emptyset \textit{ exists} \in P \text{ (L. A.4.1.13(2))} \\
 & \Rightarrow & (P, \sqsubseteq) \text{ complete lattice} \\
\text{(D. A.4.1.2 and} & & \\
\text{L. A.4.1.14)} & \Rightarrow & (P, \sqsubseteq) \text{ lattice and complete partial order} \\
\text{(D. A.4.1.1 and} & & \\
\text{D. A.3.1.1/2)} & \Rightarrow & (P, \sqsubseteq) \text{ partial order} \\
\text{(D. A.2.1.2)} & \Rightarrow & (P, \sqsubseteq) \text{ pre-order}
\end{array}
$$

Excercise A.4.1.20

Let

$$\mathcal{QO}, \mathcal{PO}, \mathcal{L}, \mathcal{CPO}, \mathcal{CL}, \mathcal{FL}, \mathcal{CJSL}, \mathcal{CJSL}_\perp, \mathcal{CMSL}, \mathcal{CMSL}^\top$$

denote the sets of all quasi-orders $\mathcal{QO}$, partial orders $\mathcal{PO}$, lattices $\mathcal{L}$, complete partial orders $\mathcal{CPO}$, complete lattices $\mathcal{CL}$, finite lattices $\mathcal{FL}$, complete join semi-lattices without/with least element $\mathcal{CJSL}/\mathcal{CJSL}_\perp$, and meet semi-lattices without/with greatest element $\mathcal{CMSL}/\mathcal{CMSL}^\top$.

1. What further implications or equivalences hold in addition to those listed in Corollary A.4.1.19? (Proof or counterexample)
2. What inclusions or (set) equalities hold among $\mathcal{QO}$, $\mathcal{PO}$, $\mathcal{L}$, etc.? (Proof or counterexample)

A.4.2

Distributive, Additive Maps on Lattices

Distributive, Additive Maps on Lattices

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a map on P.

Definition A.4.2.1 (Distributive, Additive Map)

f is called

- distributive (or $\sqcap$-continuous) iff
$$\forall\, \emptyset \neq P' \subseteq P.\ f(\bigsqcap P') = \bigsqcap f(P')$$
(Preservation of greatest lower bounds)
- additive (or $\sqcup$-continuous) iff
$$\forall\, \emptyset \neq P' \subseteq P.\ f(\bigsqcup P') = \bigsqcup f(P')$$
(Preservation of least upper bounds)

Note: $\forall\, S \subseteq P.\ f(S) =_{df} \{\, f(s) \mid s \in S \}$

Characterizing Monotonicity

...in terms of the preservation of greatest lower and least upper bounds:

Lemma A.4.2.2 (Characterizing Monotonicity)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a map on P. Then:

$$
\begin{aligned}
f \text{ is monotonic} \iff & \forall\, P' \subseteq P.\ \ f(\bigsqcap P') \sqsubseteq \bigsqcap f(P') \\
\iff & \forall\, P' \subseteq P.\ \ f(\bigsqcup P') \sqsupseteq \bigsqcup f(P')
\end{aligned}
$$

Note: $\forall\, S \subseteq P.\ f(S) =_{df} \{\, f(s) \mid s \in S \}$

Useful Results on Mon., Distr., and Additivity

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a map on P.

Lemma A.4.2.3

f is distributive iff f is additive.

Lemma A.4.2.4

f is monotonic, if f is distributive (or additive).
(i.e., distributivity (or additivity) implies monotonicity)

A.4.3

Lattice Homomorphisms, Lattice Isomorphisms

Lattice Homomorphisms, Lattice Isomorphisms

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a map from P to R.

Definition A.4.3.1 (Lattice Homorphism)

f is called a lattice homomorphism, if

$$\forall\, p, q \in P.\ f(p \sqcup_P q) = f(p) \sqcup_Q f(q) \wedge$$
$$f(p \sqcap_P q) = f(p) \sqcap_Q f(q)$$

Definition A.4.3.2 (Lattice Isomorphism)

1. f is called a lattice isomorphism, if f is a lattice homomorphism and bijective.
2. $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ are called isomorphic, if there is lattice isomorphism between P and R.

Useful Results (1)

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a map from P to R.

Lemma A.4.3.3

$$f \in [P \stackrel{hom}{\to} R] \;\Rightarrow\; f \in [P \stackrel{mon}{\to} R]$$

The reverse implication of Lemma A.4.3.3 does not hold, however, the following weaker relation holds:

Lemma A.4.3.4

$$\begin{aligned} f \in [P \stackrel{mon}{\to} R] \;\Rightarrow\; & \\ & \forall\, p, q \in P.\ f(p \sqcup_P q) \sqsupseteq_Q f(p) \sqcup_Q f(q) \;\wedge \\ & \qquad f(p \sqcap_P q) \sqsubseteq_Q f(p) \sqcap_Q f(q) \end{aligned}$$

Useful Results (2)

Let $(P, \sqsubseteq_P)$ and $(R, \sqsubseteq_R)$ be two lattices, and let $f \in [P \to R]$ be a map from P to R.

Lemma A.4.3.5

$$f \in [P \stackrel{iso}{\to} R] \;\Rightarrow\; f^{-1} \in [R \stackrel{iso}{\to} P]$$

Lemma A.4.3.6

$$f \in [P \stackrel{iso}{\to} R] \iff f \in [P \stackrel{po-hom}{\longrightarrow} R] \text{ wrt } \sqsubseteq_P \text{ and } \sqsubseteq_Q$$

A.4.4

Modular, Distributive, and Boolean Lattices

Modular Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$ and join operation $\sqcup$.

Lemma A.4.4.1

$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) \sqsubseteq (p \sqcup q) \sqcap r$$

Definition A.4.4.2 (Modular Lattice)

$(P, \sqsubseteq)$ is called modular, if

$$\forall\, p, q, r \in P.\ p \sqsubseteq r \Rightarrow p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap r$$

Characterizing Modular Lattices

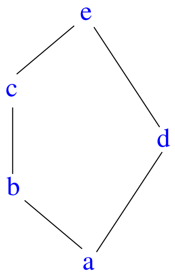
Theorem A.4.4.3 (Characterizing Modular Lattices)

A lattice $(P, \sqsubseteq)$ is

1. modular iff

   $$\forall\, p, q, r \in P.\ p \sqsubseteq q,\ p \sqcap r = q \sqcap r,\ p \sqcup r = q \sqcup r \Rightarrow p = q$$

2. not modular iff $(P, \sqsubseteq)$ contains a sublattice, which is isomorphic to the lattice:



Distributive Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$ and join operation $\sqcup$.

Lemma A.4.4.4

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) \sqsubseteq (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) \sqsupseteq (p \sqcap q) \sqcup (p \sqcap r)$

Definition A.4.4.5 (Distributive Lattice)

$(P, \sqsubseteq)$ is called distributive, if

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Lemma A.4.4.6

The following two statements are equivalent:

1. $\forall\, p, q, r \in P.\ p \sqcup (q \sqcap r) = (p \sqcup q) \sqcap (p \sqcup r)$
2. $\forall\, p, q, r \in P.\ p \sqcap (q \sqcup r) = (p \sqcap q) \sqcup (p \sqcap r)$

Hence, it is sufficient to require the validity of property (1) or of property (2) in Definition A.4.4.5.

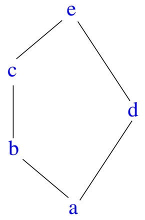
Characterizing Distributive Lattices

Let $(P, \sqsubseteq)$ be a lattice.

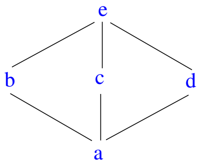
Theorem A.4.4.7 (Characterizing Distributive Lat.)

$(P, \sqsubseteq)$ is not distributive iff $(P, \sqsubseteq)$ contains a sublattice that is isomorphic to one of the below two lattices:

a)



b)



Corollary A.4.4.8

If $(P, \sqsubseteq)$ is distributive, then $(P, \sqsubseteq)$ is modular.

Boolean Lattices

Let $(P, \sqsubseteq)$ be a lattice with meet operation $\sqcap$, join operation $\sqcup$, least element $\perp$, and greatest element $\top$.

Definition A.4.4.9 (Complement)

Let $p, q \in P$. Then:

1. q is called a complement of p, if $p \sqcup q = \top$ and $p \sqcap q = \perp$.
2. P is called complementary, if every element in P has a complement.

Definition A.4.4.10 (Boolean Lattice)

$(P, \sqsubseteq)$ is called Boolean, if it is complementary, distributive, and $\perp \neq \top$.

Note: If $(P, \sqsubseteq)$ is Boolean, then every element $p \in P$ has an unambiguous unique complement in P, which is denoted by $\bar{p}$.

Useful Result

Lemma A.4.4.11

Let $(P, \sqsubseteq)$ be a Boolean lattice, and let $p, q, r \in P$. Then:

1. $\bar{\bar{p}} = p$ (Involution Law)
2. $\overline{p \sqcup q} = \bar{p} \sqcap \bar{q}, \quad \overline{p \sqcap q} = \bar{p} \sqcup \bar{q}$ (De Morgan Laws)
3. $p \sqsubseteq q \iff \bar{p} \sqcup q = \top \iff p \sqcap \bar{q} = \bot$
4. $p \sqsubseteq q \sqcup r \iff p \sqcap \bar{q} \sqsubseteq r \iff \bar{q} \sqsubseteq \bar{p} \sqcup r$

Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two Boolean lattices, and let $f \in [P \to Q]$ be a function from P to Q.

Definition A.4.4.12 (Boolean Lattice Homorphism)

f is called a Boolean lattice homomorphism, if f is a lattice homomorphism and

$$\forall\, p \in P.\; f(\bar{p}) = \overline{f(p)}$$

Definition A.4.4.13 (Boolean Lattice Isomorphism)

f is called a Boolean lattice isomorphism, if f is a Boolean lattice homomorphism and bijective.

Useful Results

Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two Boolean lattices, and let $f \in [P \overset{bhom}{\to} Q]$ be a Boolean lattice homomorphism from P to Q.

Lemma A.4.4.14

$$f(\bot) = \bot \wedge f(\top) = \top$$

Lemma A.4.4.15

f is a Boolean lattice isomorphism iff $f(\bot) = \bot \wedge f(\top) = \top$

Summary, Overview

Corollary A.4.4.16

Let $P \neq \emptyset$ be a non-empty set, and $\sqsubseteq$ a relation on P. Then:

$$\begin{array}{rcl} & & (P, \sqsubseteq) \text{ Boolean lattice} \\ (\text{Def. A.4.4.10}) & \Rightarrow & (P, \sqsubseteq) \text{ distributive lattice} \\ (\text{Cor. A.4.4.8}) & \Rightarrow & (P, \sqsubseteq) \text{ modular lattice} \\ (\text{Def. A.4.4.2}) & \Rightarrow & (P, \sqsubseteq) \text{ lattice} \\ (\text{Def. A.4.1.1}) & \Rightarrow & (P, \sqsubseteq) \text{ partial order} \\ (\text{Def. A.2.1.2}) & \Rightarrow & (P, \sqsubseteq) \text{ pre-order} \end{array}$$

Corollary A.4.4.17

$$\mathcal{QO} \supset \mathcal{PO} \supset \mathcal{L} \supset \mathcal{ML} \supset \mathcal{DL} \supset \mathcal{BL}$$

where all inclusions are proper and $\mathcal{QO}$, $\mathcal{PO}$, $\mathcal{L}$, $\mathcal{ML}$, $\mathcal{DL}$, and $\mathcal{BL}$ denote the sets of all quasi- (or pre-) orders, partial orders, lattices, modular, distributive, and Boolean lattices.

Exercise A.4.4.18

Let $(\mathbb{N}_0, \sqsubseteq)$ be the partial order with $\sqsubseteq =_{df} |$, where $|$ denotes the divisibility relation on the natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \,|\, 35$.

Prove or disprove: $(\mathbb{N}_0, \sqsubseteq)$ is a

1. modular lattice
2. distributive lattice
3. Boolean lattice

Proof or counterexample.

A.4.5

Mechanisms for Constructing Lattices

Common Lattice Constructions: Flat Lattices

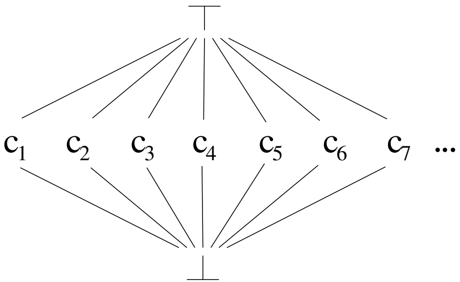
Lemma A.4.5.1 (Flat Lattice Construction)

Let C be a set. Then:

$(C \mathbin{\dot\cup} \{\bot, \top\}, \sqsubseteq_{flat})$ with $\sqsubseteq_{flat}$ defined by

$$\forall\, c, d \in C \mathbin{\dot\cup} \{\bot, \top\}.\ c \sqsubseteq_{flat} d \iff_{df} c = \bot \vee c = d \vee d = \top$$

is a complete lattice, a so-called flat lattice (or diamond lattice).



Lattice Constructions: Products, Sums,...

Like the principle for constructing flat CPOs also the principles for constructing

- non-strict products
- strict products
- separate sums
- coalesced sums
- continuous (here: additive, distributive) function spaces

carry over from CPOs to (complete) lattices (cf. App. A.3.3).

A.4.6

Order-theoretic and Algebraic View of Lattices

Motivation

In Definition A.4.1.1, we introduced lattices as special

- ordered sets $(P, \sqsubseteq)$

which induces an

- order-theoretic view of lattices.

Alternatively, lattices can be introduced as special

- algebraic structures $(P, \sqcap, \sqcup)$

which induces an

- algebraic view of lattices.

Next, we will show that both views are equivalent:

- Order-theoretically defined lattices can be considered algebraically and vice versa.

Lattices as Algebraic Structures

Definition A.4.6.1 (Algebraic Lattice)

An algebraic lattice is an algebraic structure $(P, \sqcap, \sqcup)$, where

- $P \neq \emptyset$ is a non-empty set.
- $\sqcap, \sqcup : P \times P \to P$ are two maps such that for all elements $p, q, r \in P$ the following laws hold (infix notation):
  - Commutative Laws: $p \sqcap q = q \sqcap p$
    $p \sqcup q = q \sqcup p$
  - Associative Laws: $(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$
    $(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$
  - Absorption Laws: $(p \sqcap q) \sqcup p = p$
    $(p \sqcup q) \sqcap p = p$

Properties of Algebraic Lattices

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice.

Lemma A.4.6.2 (Idempotency Laws)

For all $p \in P$, the maps $\sqcap, \sqcup : P \times P \to P$ satisfy the following laws:

- Idempotency Laws: $$\begin{aligned} p \sqcap p &= p \\ p \sqcup p &= p \end{aligned}$$

Lemma A.4.6.3

For all $p, q \in P$, the maps $\sqcap, \sqcup : P \times P \to P$ satisfy:

1. $p \sqcap q = p \iff p \sqcup q = q$
2. $p \sqcap q = p \sqcup q \iff p = q$

Induced (Partial) Order

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice.

Lemma A.4.6.4

The relation $\sqsubseteq \subseteq P \times P$ on P defined by

$$\forall\, p, q \in P.\; p \sqsubseteq q \iff_{df} p \sqcap q = p$$

is a partial order relation on P, i.e., $\sqsubseteq$ is reflexive, transitive, and antisymmetric.

Definition A.4.6.5 (Induced Partial Order)

The relation $\sqsubseteq$ defined in Lemma A.4.6.4 is called the induced (partial) order of $(P, \sqcap, \sqcup)$.

Properties of the Induced Partial Order

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice, and let $\sqsubseteq$ be the induced partial order of $(P, \sqcap, \sqcup)$.

Lemma A.4.6.6

For all $p, q \in P$, the infimum ($\widehat{=}$ greatest lower bound) and the supremum ($\widehat{=}$ least upper bound) of the set $\{p, q\}$ exist and are given by the images of $\sqcap$ and $\sqcup$ applied to p and q, respectively, i.e.:

$$\forall\, p, q \in P.\ \bigsqcap\{p, q\} = p \sqcap q \ \wedge\ \bigsqcup\{p, q\} = p \sqcup q$$

Lemma A.4.6.6 can inductively be extended yielding:

Lemma A.4.6.7

Let $\emptyset \neq Q \subseteq P$ be a non-empty finite subset of P. Then:

$$\exists\, glb, lub \in P.\ glb = \bigsqcap Q \ \wedge\ lub = \bigsqcup Q$$

Algebraic Lattices Order-theoretically

Corollary A.4.6.8 (From $(P, \sqcap, \sqcup)$ to $(P, \sqsubseteq)$)

Let $(P, \sqcap, \sqcup)$ be an algebraic lattice. Then:

$(P, \sqsubseteq)$, where $\sqsubseteq$ is the induced partial order of $(P, \sqcap, \sqcup)$, is an order-theoretic lattice in the sense of Definition A.4.1.1.

Induced Algebraic Maps

Let $(P, \sqsubseteq)$ be an order-theoretic lattice.

Definition A.4.6.9 (Induced Algebraic Maps)

The partial order $\sqsubseteq$ of $(P, \sqsubseteq)$ induces two maps $\sqcap$ and $\sqcup$ from $P \times P$ to P defined by:

1. $\forall\, p, q \in P.\ p \sqcap q =_{df} \bigsqcap\{p, q\}$
2. $\forall\, p, q \in P.\ p \sqcup q =_{df} \bigsqcup\{p, q\}$

Properties of the Induced Algebraic Maps (1)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice, and let $\sqcap$ and $\sqcup$ be the induced algebraic maps of $(P, \sqsubseteq)$.

Lemma A.4.6.10

Let $p, q \in P$. Then the following statements are equivalent:

1. $p \sqsubseteq q$
2. $p \sqcap q = p$
3. $p \sqcup q = q$

Properties of the Induced Algebraic Maps (2)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice, and let $\sqcap$ and $\sqcup$ be the induced algebraic maps of $(P, \sqsubseteq)$.

Lemma A.4.6.11

For all $p, q, r \in P$, the induced maps $\sqcap$ and $\sqcup$ satisfy the following laws:

- Commutative Laws: $$p \sqcap q = q \sqcap p$$ $$p \sqcup q = q \sqcup p$$
- Associative Laws: $$(p \sqcap q) \sqcap r = p \sqcap (q \sqcap r)$$ $$(p \sqcup q) \sqcup r = p \sqcup (q \sqcup r)$$
- Absorption Laws: $$(p \sqcap q) \sqcup p = p$$ $$(p \sqcup q) \sqcap p = p$$
- Idempotency Laws: $$p \sqcap p = p$$ $$p \sqcup p = p$$

Order-theoretic Lattices Algebraically

Corollary A.4.6.12 (From $(P, \sqsubseteq)$ to $(P, \sqcap, \sqcup)$)

Let $(P, \sqsubseteq)$ be an order-theoretic lattice. Then:

$(P, \sqcap, \sqcup)$, where $\sqcap$ and $\sqcup$ are the induced maps of $(P, \sqsubseteq)$, is an algebraic lattice in the sense of Definition A.4.6.1.

Equivalence (1)

...of the order-theoretic and the algebraic view of lattices.

From order-theoretic to algebraic lattices:

- An order-theoretic lattice $(P, \sqsubseteq)$ can be considered algebraically by switching from $(P, \sqsubseteq)$ to $(P, \sqcap, \sqcup)$, where $\sqcap$ and $\sqcup$ are the induced maps of $(P, \sqsubseteq)$.

From algebraic to order-theoretic lattices:

- An algebraic lattice $(P, \sqcap, \sqcup)$ can be considered order-theoretically by switching from $(P, \sqcap, \sqcup)$ to $(P, \sqsubseteq)$, where $\sqsubseteq$ is the induced partial order of $(P, \sqcap, \sqcup)$.

Equivalence (2)

Together, this allows us to simply speak of a lattice P, and to speak only more precisely of P as an

- order-theoretic lattice $(P, \sqsubseteq)$
- algebraic lattice $(P, \sqcap, \sqcup)$

if we want to emphasize that we think of P as a special ordered set or as a special algebraic structure.

Bottom and Top vs. Zero and One (1)

Let P be a lattice with a least and a greatest element.

Considering P

- order-theoretically as $(P, \sqsubseteq)$, it is appropriate to think of its least and greatest element in terms of bottom $\perp$ and top $\top$ with
  - Bottom $\perp \in P$: $\perp = \bigsqcup \emptyset$
  - Top $\top \in P$: $\top = \bigsqcap \emptyset$
- algebraically as $(P, \sqcap, \sqcup)$, it is appropriate to think of its least and greatest element in terms of Zero $\mathbf{0}$ and One $\mathbf{1}$, where $(P, \sqcap, \sqcup)$ is said to have a (if existent, uniquely determined)
  - Zero $\mathbf{0} \in P$: $\forall\, p \in P.\ p \sqcup \mathbf{0} = p$
  - One $\mathbf{1} \in P$: $\forall\, p \in P.\ p \sqcap \mathbf{1} = p$

Bottom and Top vs. Zero and One (2)

Lemma A.4.6.13

Let P be a lattice. Then:

- $(P, \sqsubseteq)$ has a bottom element $\perp$ iff $(P, \sqcap, \sqcup)$ has a zero element $\mathbf{0}$, and in that case:

$$(\bigsqcup \emptyset =)\ \perp = \mathbf{0}$$

- $(P, \sqsubseteq)$ has a top element $\top$ iff $(P, \sqcap, \sqcup)$ has a one element $\mathbf{1}$, and in that case:

$$(\bigsqcap \emptyset =)\ \top = \mathbf{1}$$

On the Adequacy of the two Lattice Views

In mathematics, usually the

- algebraic view of a lattice is more appropriate as it is in line with other algebraic structures ('a set together with some maps satisfying a number of laws'), e.g., groups, rings, fields, vector spaces, categories, etc., which are investigated and dealt with in mathematics.

In computer science, usually the

- order-theoretic view of a lattice is more appropriate, since the order relation can often be interpreted and understood as '· carries more/less information than ·,' '· is more/less defined than ·,' '· is stronger/weaker than ·,' etc., which often fits naturally to problems investigated and dealt with in computer science.

Exercise A.4.6.14

Let $(\mathbb{N}_0, \sqsubseteq)$ be the lattice with $\sqsubseteq =_{df} \mid$, where $\mid$ denotes the divisibility relation on the natural numbers $\mathbb{N}_0$, i.e., the relation '$\cdot$ divides $\cdot$' (w/out remainder), e.g. $5 \mid 35$.

Provide the definition of $(\mathbb{N}_0, \wedge, \vee)$, i.e., write down the algebraically defined counterpart of $(\mathbb{N}_0, \sqsubseteq)$. To this end, provide the definition of the meet and join operation on $\mathbb{N}_0 \times \mathbb{N}_0$:

1. $\wedge : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$
2. $\vee : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$

What is the

1. zero element $\mathbf{0}$
2. one element $\mathbf{1}$

of $(\mathbb{N}_0, \wedge, \vee)$?

A.5

Fixed Point Theorems

A.5.1

Fixed Points, Towers

Fixed Points of Functions

Definition A.5.1.1 (Fixed Point)

Let M be a set, let $f \in [M \to M]$ be a function on M, and let $m \in M$ be an element of M. Then:

m is called a fixed point of f iff $f(m) = m$.

Least, Greatest Fixed Points in Partial Orders

Definition A.5.1.2 (Least, Greatest Fixed Point)

Let $(P, \sqsubseteq)$ be a partial order, let $f \in [P \to P]$ be a function on P, and let p be a fixed point of f, i.e., $f(p) = p$. Then:

p is called the

- least fixed point of f, denoted by μf,
  iff $\forall\, q \in P.\ f(q) = q \Rightarrow p \sqsubseteq q$
- greatest fixed point of f, denoted by νf,
  iff $\forall\, q \in P.\ f(q) = q \Rightarrow q \sqsubseteq p$

Towers in Chain Complete Partial Orders

Definition A.5.1.3 (f-Tower in C)

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \to C]$ be a function on C, and let $T \subseteq C$ be a subset of C. Then:

T is called an f-tower in C iff

1. $\perp \in T$.
2. If $t \in T$, then also $f(t) \in T$.
3. If $T' \subseteq T$ is a chain in C, then $\bigsqcup T' \in T$.

Least Towers in Chain Complete Partial Orders

Lemma A.5.1.4 (The Least f-Tower in C)

The intersection

$$I =_{df} \bigcap \{ T \mid T \ f\text{-tower in } C \}$$

of all f-towers in C is the least f-tower in C, i.e.,

1. I is an f-tower in C.
2. $\forall \, T \ f$-tower in C. $I \subseteq T$.

Lemma A.5.1.5 (Least f-Towers and Chains)

The least f-tower in C is a chain in C, if f is expanding.

A.5.2

Fixed Point Theorems for Complete Partial Orders

Fixed Points of Exp./Monotonic Functions

Fixed Point Theorem A.5.2.1 (Expanding Function)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \stackrel{exp}{\to} C]$ be an expanding function on C. Then:

The supremum of the least f-tower in C is a fixed point of f.

Fixed Point Theorem A.5.2.2 (Monotonic Function)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \stackrel{mon}{\to} C]$ be a monotonic function on C. Then:

f has a unique least fixed point μf, which is given by the supremum of the least f-tower in C.

Note

- Theorem A.5.2.1 and Theorem A.5.2.2 ensure the existence of a fixed point for expanding functions and of a unique least fixed point for monotonic functions, respectively, but do not provide constructive procedures for computing or approximating them.
- This is in contrast to Theorem A.5.2.3, which does so for continuous functions. In practice, continuous functions are thus more important and considered where possible.

Least Fixed Points of Continuous Functions

Fixed Point Theorem A.5.2.3 (Knaster,Tarski,Kleene)

Let $(C, \sqsubseteq)$ be a CCPO, and let $f \in [C \overset{con}{\to} C]$ be a continuous function on C. Then:

f has a unique least fixed point $\mu f \in C$, which is given by the supremum of the (so-called) Kleene chain $\{\bot, f(\bot), f^2(\bot), \ldots\}$, i.e.:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{\bot, f(\bot), f^2(\bot), \ldots\}$$

Note: $f^0 =_{df} Id_C$; $f^i =_{df} f \circ f^{i-1}$, $i > 0$.

Proof of Fixed Point Theorem A.5.2.3 (1)

We have to prove:

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) = \bigsqcup \{ f^i(\bot) \mid i \geq 0 \}$$

1. exists,
2. is a fixed point of f,
3. is the least fixed point of f.

Proof of Fixed Point Theorem A.5.2.3 (2)

1. Existence

- By definition of $\perp$ as the least element of C and of f^0 as the identity on C we have: $\perp = f^0(\perp) \sqsubseteq f^1(\perp) = f(\perp)$.
- Since f is continuous and hence monotonic, we obtain by means of (natural) induction:
  $\forall\, i, j \in \mathbb{N}_0.\ i < j \Rightarrow f^i(\perp) \sqsubseteq f^{i+1}(\perp) \sqsubseteq f^j(\perp)$.
- Hence, the set $\{f^i(\perp) \mid i \geq 0\}$ is a (possibly infinite) chain in C.
- Since $(C, \sqsubseteq)$ is a CCPO and $\{f^i(\perp) \mid i \geq 0\}$ a chain in C, this implies by definition of a CCPO that the least upper bound of the chain $\{f^i(\perp) \mid i \geq 0\}$

$$\bigsqcup \{f^i(\perp) \mid i \geq 0\} = \bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) \text{ exists.}$$

Proof of Fixed Point Theorem A.5.2.3 (3)

2. Fixed point property

$$
\begin{array}{rcl}
 & & f(\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)) \\
(f \text{ continuous}) & = & \bigsqcup_{i \in \mathbb{N}_0} f(f^i(\bot)) \\
 & = & \bigsqcup_{i \in \mathbb{N}_1} f^i(\bot) \\
(C' =_{df} \{f^i \bot \mid i \geq 1\} \text{ is a chain} \Rightarrow & & \\
\bigsqcup C' \text{ exists } = \bot \sqcup \bigsqcup C') & = & \bot \sqcup \bigsqcup_{i \in \mathbb{N}_1} f^i(\bot) \\
(f^0(\bot) =_{df} \bot) & = & \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)
\end{array}
$$

Proof of Fixed Point Theorem A.5.2.3 (4)

3. Least fixed point property

- Let c be an arbitrary fixed point of f. Then: $\perp \sqsubseteq c$.
- Since f is continuous and hence monotonic, we obtain by means of (natural) induction:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\perp) \sqsubseteq f^i(c)\ (= c)$.
- Since c is a fixed point of f, this implies:
  $\forall\, i \in \mathbb{N}_0.\ f^i(\perp) \sqsubseteq c\ (= f^i(c))$.
- Thus, c is an upper bound of the set $\{f^i(\perp) \mid i \in \mathbb{N}_0\}$.
- Since $\{f^i(\perp) \mid i \in \mathbb{N}_0\}$ is a chain, and $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)$ is by definition the least upper bound of this chain, we obtain the desired inclusion

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) \sqsubseteq c.$$

□

Least Conditional Fixed Points

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \to C]$ be a function on C, and let $d, c_d \in C$ be elements of C.

Definition A.5.2.4 (Least Conditional Fixed Point)

c_d is called the least conditional fixed point of f wrt d (in German: kleinster bedingter Fixpunkt) iff c_d is the least fixed point of C with $d \sqsubseteq c_d$, i.e.:

$$\forall\, x \in C.\ f(x) = x \ \wedge\ d \sqsubseteq x \Rightarrow c_d \sqsubseteq x$$

Least Cond. Fixed Points of Cont. Functions

Theorem A.5.2.5 (Conditional Fixed Point Theorem)

Let $(C, \sqsubseteq)$ be a CCPO, let $d \in C$, and let $f \in [C \stackrel{con}{\to} C]$ be a continuous function on C which is expanding for d, i.e., $d \sqsubseteq f(d)$. Then:

f has a least conditional fixed point $\mu f_d \in C$, which is given by the supremum of the (generalized) Kleene chain $\{d, f(d), f^2(d), \ldots\}$, i.e.:

$$\mu f_d = \bigsqcup_{i \in \mathbb{N}_0} f^i(d) = \bigsqcup \{d, f(d), f^2(d), \ldots\}$$

Finite Fixed Points

Let $(C, \sqsubseteq)$ be a CCPO, let $d \in C$, and let $f \in [C \overset{mon}{\to} C]$ be a monotonic function on C.

Theorem A.5.2.6 (Finite Fixed Point Theorem)

If two succeeding elements in the Kleene chain of f are equal, i.e., if there is some $i \in \mathbb{N}$ with $f^i(\perp) = f^{i+1}(\perp)$, then we have: $\mu f = f^i(\perp)$.

Theorem A.5.2.7 (Finite Conditional FP Theorem)

If f is expanding for d, i.e., $d \sqsubseteq f(d)$, and two succeeding elements in the (generalized) Kleene chain of f wrt d are equal, i.e., if there is some $i \in \mathbb{N}$ with $f^i(d) = f^{i+1}(d)$, then we have: $\mu f_d = f^i(d)$.

Note: Theorems A.5.2.6 and A.5.2.7 do not require continuity of f. Monotonicity (and expandingness) of f suffice(s).

Towards the Existence of Finite Fixed Points

Let $(P, \sqsubseteq)$ be a partial order, and let $p, r \in P$.

Definition A.5.2.8 (Chain-finite Partial Order)

$(P, \sqsubseteq)$ is called chain-finite (in German: kettenendlich) iff P does not contain an infinite chain.

Definition A.5.2.9 (Finite Element)

p is called

- finite iff the set $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$ does not contain an infinite chain.
- finite relative to r iff the set $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$ does not contain an infinite chain.

Existence of Finite Fixed Points

...there are numerous sufficient conditions ensuring the existence of a least finite fixed point of a function f, which often hold in practice (cf. Nielson/Nielson 1992), e.g.:

- the domain or the range of f are finite or chain-finite,
- the least fixed point of f is finite,
- f is of the form $f(c) = c \sqcup g(c)$ with g a monotonic function on a chain-finite (data) domain.

Fixed Point Theorems, Lattices, and DCPOs

Note: Complete lattices (cf. Lemma A.4.1.13) and DCPOs with a least element (cf. Lemma A.3.1.5) are CCPOs, too.

Thus, we can conclude:

Corollary A.5.2.10 (Fixed Points, Lattices, DCPOs)

The fixed point theorems of Chapter A.5.2 hold for functions on complete lattices and on DCPOs with a least element, too.

A.5.3

Fixed Point Theorems for Lattices

Fixed Points of Monotonic Functions

Fixed Point Theorem A.5.3.1 (Knaster, Tarski)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \stackrel{mon}{\to} P]$ be a monotonic function on P. Then:

1. f has a unique least fixed point $\mu f \in P$, which is given by $\mu f = \bigsqcap\{p \in P \mid f(p) \sqsubseteq p\}$.
2. f has a unique greatest fixed point $\nu f \in P$, which is given by $\nu f = \bigsqcup\{p \in P \mid p \sqsubseteq f(p)\}$.

Characterization Theorem A.5.3.2 (Davis)

Let $(P, \sqsubseteq)$ be a lattice. Then:

$(P, \sqsubseteq)$ is complete iff every $f \in [P \stackrel{mon}{\to} P]$ has a fixed point.

The Fixed Point Lattice of Mon. Functions

Theorem A.5.3.3 (Lattice of Fixed Points)

Let $(P, \sqsubseteq)$ be a complete lattice, let $f \in [P \overset{mon}{\to} P]$ be a monotonic function on P, and let $Fix(f) =_{df} \{p \in P \mid f(p) = p\}$ be the set of all fixed points of f. Then:

Every subset $F \subseteq Fix(f)$ has a supremum and an infimum in $Fix(f)$, i.e., $(Fix(f), \sqsubseteq_{|Fix(f)})$ is a complete lattice.

Theorem A.5.3.4 (Ordering of Fixed Points)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \overset{mon}{\to} P]$ be a monotonic function on P. Then:

$$\bigsqcup_{i \in \mathbb{N}_0} f^i(\bot) \sqsubseteq \mu f \sqsubseteq \nu f \sqsubseteq \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$$

Fixed Points of Add./Distributive Functions

For additive and distributive functions, the leftmost and the rightmost inequality of Theorem A.5.3.4 become equalities:

Fixed Point Theorem A.5.3.5 (Knaster,Tarski,Kleene)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $f \in [P \to P]$ be a function on P. Then: f has a unique

1. least fixed point $\mu f \in P$ given by $\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\bot)$, if f is additive, i.e., $f \in [P \stackrel{add}{\to} P]$.
2. greatest fixed point $\nu f \in P$ given by $\nu f = \bigsqcap_{i \in \mathbb{N}_0} f^i(\top)$, if f is distributive, i.e., $f \in [P \stackrel{dis}{\to} P]$.

Recall: $f^0 =_{df} Id_C$; $f^i =_{df} f \circ f^{i-1}$, $i > 0$.

A.6

Fixed Point Induction

Admissible Predicates

Fixed point induction allows proving properties of fixed points. Essential is the notion of admissible predicates:

Definition A.6.1 (Admissible Predicate)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be a predicate on P. Then:

ϕ is called admissible (or $\sqcup$-admissible) iff for every chain $C \subseteq P$ holds:

$$(\forall\, c \in C.\ \phi(c)) \Rightarrow \phi(\bigsqcup C)$$

Lemma A.6.2

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be an admissible predicate on P. Then: $\phi(\bot) = \textbf{wahr}$.

Proof. The admissibility of ϕ implies $\phi(\bigsqcup \emptyset) = \textbf{wahr}$. Moreover, we have $\bot = \bigsqcup \emptyset$, which completes the proof.

Sufficient Conditions for Admissibility

Theorem A.6.3 (Admissibility Condition 1)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi : P \to \mathbb{B}$ be a predicate on P. Then:

ϕ is admissible, if there is a complete lattice $(Q, \sqsubseteq_Q)$ and two additive functions $f, g \in [P \stackrel{add}{\to} Q]$, such that

$$\forall\, p \in P.\ \phi(p) \iff f(p) \sqsubseteq_Q g(p)$$

Theorem A.6.4 (Admissibility Condition 2)

Let $(P, \sqsubseteq)$ be a complete lattice, and let $\phi, \psi : P \to \mathbb{B}$ be two admissible predicates on P. Then:

The conjunction of ϕ and ψ, the predicate $\phi \ \wedge\ \psi$ defined by

$$\forall\, p \in P.\ (\phi \ \wedge\ \psi)(p) =_{df} \phi(p) \wedge \psi(p)$$

is admissible.

Fixed Point Induction on Complete Lattices

Theorem A.6.5 (Fixed Point Induction on C. Lat.)

Let $(P, \sqsubseteq)$ be a complete lattice, let $f \in [P \stackrel{add}{\to} P]$ be an additive function on P, and let $\phi : P \to \mathbb{B}$ be an admissible predicate on P. Then:

The validity of

- $\forall\, p \in P.\ \phi(p) \Rightarrow \phi(f(p))$ (Induction step)

implies the validity of $\phi(\mu f)$.

Note: The induction base, i.e., the validity of $\phi(\perp)$, is implied by the admissibility of ϕ (cf. Lemma A.6.2) and proved when verifying the admissibility of ϕ.

Fixed Point Induction on CCPOs

The notion of admissibility of a predicate carries over from complete lattices to CCPOs.

Theorem A.6.6 (Fixed Point Induction on CCPOs)

Let $(C, \sqsubseteq)$ be a CCPO, let $f \in [C \stackrel{mon}{\to} C]$ be a monotonic function on C, and let $\phi : C \to \mathbb{B}$ be an admissible predicate on C. Then:

The validity of

- $\forall\, c \in C.\ \phi(c) \Rightarrow \phi(f(c))$ (Induction step)

implies the validity of $\phi(\mu f)$.

Note: Theorem A.6.6 holds (of course still), if we replace the CCPO $(C, \sqsubseteq)$ by a complete lattice $(P, \sqsubseteq)$.

A.7

References, Further Reading

Appendix A: Further Reading (1)

- André Arnold, Irène Guessarian. *Mathematics for Computer Science*. Prentice Hall, 1996.
- Roland Backhouse, Roy Crole, Jeremy R. Gibbons (Eds.). *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*. International Summer School and Workshop, Oxford, UK, April 10-14, 2000, Revised Lectures, Springer-V., LNCS 2297, 2002. (Chapter 1, Ordered Sets and Complete Lattices by Hilary A. Priestley; Chapter 2, Algebras and Coalgebras by Peter Aczel; Chapter 4, Calculating Functional Programs by Jeremy Gibbons)
- Rudolf Berghammer. *Ordnungen, Verbände und Relationen mit Anwendungen*. Vieweg+Teubner, 2008.

Appendix A: Further Reading (2)

- Rudolf Berghammer. *Ordnungen, Verbände und Relationen mit Anwendungen*. Springer-V., 2012. (Kapitel 1, Ordnungen und Verbände; Kapitel 2.4, Vollständige Verbände; Kapitel 3, Fixpunkttheorie mit Anwendungen; Kapitel 4, Vervollständigung und Darstellung mittels Vervollständigung; Kapitel 5, Wohlgeordnete Mengen und das Auswahlaxiom)

- Rudolf Berghammer. *Ordnungen und Verbände: Grundlagen, Vorgehensweisen und Anwendungen*. Springer-V., 2013. (Kapitel 2, Verbände und Ordnungen; Kapitel 3.4, Vollständige Verbände; Kapitel 4, Fixpunkttheorie mit Anwendungen; Kapitel 5, Vervollständigung und Darstellung mittels Vervollständigung; Kapitel 6, Wohlgeordnete Mengen und das Auswahlaxiom)

Appendix A: Further Reading (3)

- Garret Birkhoff. *Applications of Lattice Algebra*. Mathematical Proceedings of the Cambridge Philosophical Society 30(2):115-122, 1934.
- Garret Birkhoff. *Lattice Theory*. American Mathematical Society, 3rd edition, 1967.
- Peter Crawley, Robert P. Dilworth. *Algebraic Theory of Lattices*. Prentice Hall, 1973.
- Brian A. Davey, Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks, Cambridge University Press, 2nd edition, 2002. (Chapter 1, Ordered Sets; Chapter 2, Lattices and Complete Lattices; Chapter 8, CPOs and Fixpoint Theorems)

Appendix A: Further Reading (4)

- Anne C. Davis. *A Characterization of Complete Lattices*. Pacific Journal of Mathematics 5(2):311-319, 1955.
- Marcel Erné. *Einführung in die Ordnungstheorie*. Bibliographisches Institut, 2. Auflage, 1982.
- Helmuth Gericke. *Theorie der Verbände*. Bibliographisches Institut, 2. Auflage, 1967.
- George Grätzer. *General Lattice Theory*. Birkhäuser, 2nd edition, 1998. (Chapter 1, First Concepts; Chapter 2, Distributive Lattices; Chapter 3, Congruences and Ideals; Chapter 5, Varieties of Lattices)
- George Grätzer. *Lattice Theory: Foundation*. Birkhäuser, 2011.

Appendix A: Further Reading (5)

- George Grätzer, Friedrich Wehrung (Eds.). *Lattice Theory: Special Topics and Applications, Vol. I*. Birkhäuser, 2014.
- George Grätzer, Friedrich Wehrung (Eds.). *Lattice Theory: Special Topics and Applications, Vol. II*. Birkhäuser, 2016.
- Paul R. Halmos. *Naive Set Theory*. Springer-V., Reprint, 2001. (Chapter 6, Ordered Pairs; Chapter 7, Relations; Chapter 8, Functions)
- Hans Hermes. *Einführung in die Verbandstheorie*. Springer-V., 2. Auflage, 1967.
- Richard Johnsonbaugh. *Discrete Mathematics*. Pearson, 7th edition, 2009. (Chapter 3, Functions, Sequences, and Relations)

Appendix A: Further Reading (6)

- Stephen C. Kleene. *Introduction to Metamathematics*. North Holland, 1952. (Reprint, North Holland, 1980)
- Seymour Lipschutz. *Set Theory and Related Topics*. McGraw Hill Schaum's Outline Series, 2nd edition, 1998. (Chapter 4, Functions; Chapter 6, Relations)
- David Makinson. *Sets, Logic and Maths for Computing*. Springer-V., 2008. (Chapter 1, Collecting Things Together: Sets; Chapter 2, Comparing Things: Relations)
- George Markowsky. *Chain-complete Posets and Directed Sets with Applications*. Algebra Universalis 6(1):53-68, 1976.

Appendix A: Further Reading (7)

- Flemming Nielson, Hanne Riis Nielson. *Finiteness Conditions for Fixed Point Iteration*. In Proceedings of the 7th ACM Conference on LISP and Functional Programming (LFP'92), 96-108, 1992.
- Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 4, Denotational Semantics)
- Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 5, Denotational Semantics)

Appendix A: Further Reading (8)

- Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. Springer-V., 2nd edition, 2005. (Appendix A, Partially Ordered Sets)
- Steven Roman. *Lattices and Ordered Sets*. Springer-V., 2008.
- Bernhard Steffen, Oliver Rüthing, Malte Isberner. *Grundlagen der höheren Informatik. Induktives Vorgehen*. Springer-V., 2014. (Kapitel 5.1, Ordnungsrelationen; Kapitel 5.2, Ordnungen und Teilstrukturen)

Appendix A: Further Reading (9)

- Bernhard Steffen, Oliver Rüthing, Michael Huth. *Mathematical Foundations of Advanced Informatics: Inductive Approaches*. Springer-V., 2018. (Chapter 5.1, Order Relations; Chapter 5.2, Orders and Substructures)
- Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5(2):285-309, 1955.
- Franklyn Turbak, David Gifford with Mark A. Sheldon. *Design Concepts in Programming Languages*. MIT Press, 2008. (Chapter 5, Fixed Points; Chapter 105, Software Testing; Chapter 106, Formal Methods; Chapter 107, Verification and Validation)

Appendix B

Pragmatics: Variants of Flow Graphs

B.1

Motivation

B.1.1

Flow Graph Variants

Representing Instructions in Flow Graphs

...representing programs by flow graphs, instructions (assignments, tests) can be attached to:

- nodes
- edges

as

- single instructions
- basic blocks (i.e., sequential sequences of instructions)

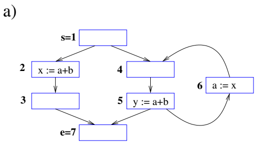
Flow Graph Variants

This leads to four flow graph variants:

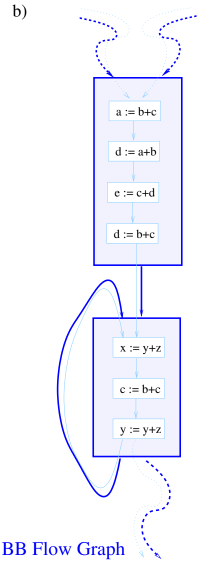
- Node-labelled flow graphs
  (in the style of Kripke structures)
  1) Single instruction graphs (SI graphs)
  2) Basic block graphs (BB graphs)
- Edge-labelled flow graphs
  (in the style of transition systems)
  3) Single instruction graphs (SI graphs)
  4) Basic block graphs (BB graphs)

Node-labelled Flow Graph Variants

a) Single instruction vs. b) basic block flow graphs:



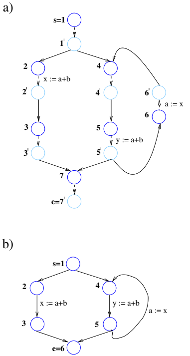
Node−labelled SI Flow Graph



Node−labelled BB Flow Graph

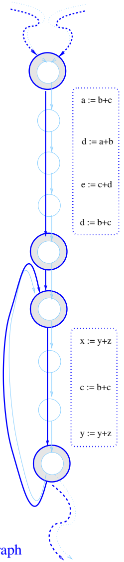
Edge-labelled Flow Graph Variants

a), b) Single instruction vs. c) basic block flow graphs:



Edge−labelled SI Flow Graphs

c)



Edge−labelled BB Flow Graph

Which Flow Graph Variant shall We Select?

Conceptually, there is

- no difference between the various flow graph variants making the choice of a particular one essentially a matter of taste.

Pragmatically, however,

- the flow graph variants differ in the ease and hence adequacy of use for specifying and implementing program analyses and optimizations.

This will be considered in more detail next.

B.1.2

Flow Graph Variants: Which One to Select?

Basic Block or Single Instruction Graphs

...node or edge-labelled, these are the questions.

We will investigate these questions by comparing the adequacy of different flow graph variants for program analysis and optimization.

To this end we consider node and edge-labelled flow graphs annototated with basic blocks and single instructions, respectively, and investigate their

- advantages and disadvantages for program analysis

from a pragmatical perspective addressing thereby especially the question:

- BB or SI graphs: Just a matter of taste?

On the fly we will learn some new data flow analyses such as

- Faint variable analysis, example of a non-separable real world data flow analysis problem.

Basic Block Graphs: Expected Advantages

Advantages commonly attributed to basic block graphs by 'folk knowledge:'

Better scalability and performance because

- less nodes (edges) are involved in the (potentially) computationally costly fixed point iteration
- larger programs fit into the main memory.

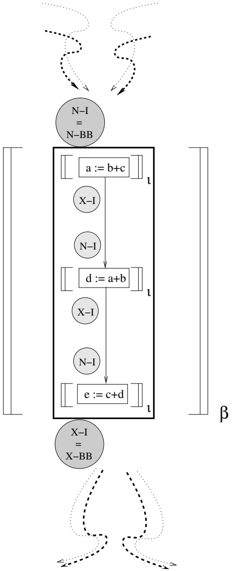
Basic Block Graphs: Definite Disadvantages

Definite disadvantages of basic block graphs in practice:

- Higher conceptual complexity: Basic blocks introduce an undesired hierarchy into flow graphs making both theoretical reasoning and practical implementations more difficult.
- Need for pre- and post-processes: These are usually required in order to cope with the additional problems introduced by the hierarchical structure of basic block flow graphs (e.g., in dead code elimination, constant propagation,...); or which necessitate 'tricky' formulations to avoid them (e.g., in partial redundancy elimination).
- Limited generality: Some practically relevant program analyses and optimizations are difficult or not at all expressible on the level of basic block flow graphs (e.g., faint variable elimination).

Core Issue

...basic blocks cause a hierarchical graph structure:



In the following

...we oppose advantages and disadvantages of

- basic block (BB) and single instructions (SI) graphs

considering DFA problems already discussed:

- Available expressions
- Simple constants

and new DFA problems:

- Faint variables

B.2

MOP and *MaxFP* Approach

B.2.1

Edge-labelled Instruction Graphs

Fixing the Setting

Let

- $G = (N, E, \mathbf{s}, \mathbf{e})$ be an edge-labelled SI flow graph.
- $\mathcal{S}_G =_{df} (\widehat{\mathcal{C}}, [\![\ ]\!]_{E,\iota}, c_{\mathbf{s}}, fw)$ be a DFA specification.

The *MOP* Approach and *MOP* Solution

...for an edge-labelled single instruction flow graph.

Definition B.2.1.1 (The *MOP* Solution)

The $MOP_{E,\iota}$ solution of $\mathcal{S}_G$ is defined by:

$$MOP_{E,\iota}^{\mathcal{S}_G} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ MOP_{E,\iota}^{\mathcal{S}_G}(n) =_{df} \bigsqcap \{ [\![\, p \,]\!]_{E,\iota}(c_{\mathbf{s}}) \mid p \in \mathbf{P}[\mathbf{s}, n] \}$$

The *MaxFP* Approach and *MaxFP* Solution

...for an edge-labelled single instruction flow graph.

Definition B.2.1.2 (The *MaxFP* Solution)

The $\mathit{MaxFP}_{E,\iota}$ solution of $\mathcal{S}_G$ is defined by:

$$\mathit{MaxFP}^{\mathcal{S}_G}_{E,\iota} : N \to \mathcal{C}$$

$$\forall\, n \in N.\ \mathit{MaxFP}^{\mathcal{S}_G}_{E,\iota}(n) =_{df} \nu\text{-}\mathit{inf}_{c_{\mathbf{s}}}(n)$$

where $\nu\text{-}\mathit{inf}_{c_{\mathbf{s}}}$ denotes the greatest solution of the *MaxFP* Equation System for instruction graphs:

$$\mathit{inf}(n) = \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigsqcap \{\, [\![\, (m,n) \,]\!]_{E,\iota}(\mathit{inf}(m)) \mid m \in \mathit{pred}(n) \,\} & \text{otherwise} \end{cases}$$

B.2.2

Node-labelled Basic Block Graphs

Fixing the Setting (1)

In the following we denote:

- basic block nodes by boldface letters ($\mathbf{m}$, $\mathbf{n}$,...)
- single instruction nodes by normalface letters (m, n,...)

We start from:

- $G = (N, E, \mathbf{s}, \mathbf{e})$, a node-labelled SI flow graph
- $\mathcal{S}_G = (\widehat{\mathcal{C}}, [\![\ ]\!]_{N,\iota}, c_{\mathbf{s}}, fw)$, a DFA specification

which induce a node-labelled BB flow graph $\mathbf{G}$ and a corresponding DFA specification $\mathcal{S}_{\mathbf{G}}$.

Fixing the Setting (2)

Given G and $\mathcal{S}_G$, let

- $\mathbf{G} = (\mathbf{N}, \mathbf{E}, \mathbf{s_G}, \mathbf{e_G})$
- $\mathcal{S}_\mathbf{G} = (\widehat{\mathcal{C}}, [\![\ ]\!]_{\mathbf{N},\beta}, c_\mathbf{s}, fw)$

denote the node-labelled BB flow graph and the DFA specification induced by G and $\mathcal{S}_G$, respectively, where

- $[\![\ ]\!]_{\mathbf{N},\beta} : \mathbf{N} \to \mathcal{C} \to \mathcal{C}$

denotes the extension of the SI DFA functional $[\![\ ]\!]_{N,\iota}$ from nodes to basic blocks defined by

- $\forall\, \mathbf{n} = \langle n_{\iota_1}, \ldots, n_{\iota_k} \rangle \in \mathbf{N}.\ [\![\, \mathbf{n}\,]\!]_{N,\beta} =_{df} [\![\, \langle n_1, \ldots, n_k \rangle\,]\!]_{N,\iota}$

Auxiliary Mappings

- *bb*: maps a node n to the basic block $\mathbf{n}$ it is included in.
- *start*: maps a basic block node $\mathbf{n}$ to its entry node n.
- *end*: maps a basic block node $\mathbf{n}$ to its exit node n.

The *MOP* Approach and *MOP* Solution (1)

...for a node-labelled basic block flow graph.

Definition B.2.2.1 (The *MOP* Solution, Part 1)

The $MOP_{\mathbf{N},\beta}$ solution of $\mathcal{S}_{\mathbf{G}}$ is defined by

$$MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}} : \mathbf{N} \to (\mathcal{C}, \mathcal{C})$$

$$\forall\, \mathbf{n} \in \mathbf{N}.\ MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}}(\mathbf{n}) =_{df} (\, N\text{-}MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}}(\mathbf{n}),\ X\text{-}MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}}(\mathbf{n})\,)$$

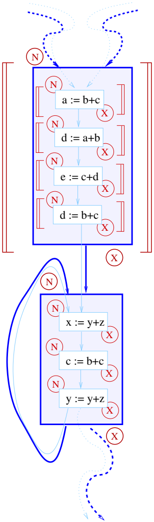
where

$$N\text{-}MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\, p \,]\!]_{\mathbf{N},\beta}(c_{\mathbf{s}}) \mid p \in \mathbf{P}_{\mathbf{G}}[\mathbf{s}, \mathbf{n}[\,\}$$

$$X\text{-}MOP_{\mathbf{N},\beta}^{\mathcal{S}_{\mathbf{G}}}(\mathbf{n}) =_{df} \bigsqcap \{\, [\![\, p \,]\!]_{\mathbf{N},\beta}(c_{\mathbf{s}}) \mid p \in \mathbf{P}_{\mathbf{G}}[\mathbf{s}, \mathbf{n}]\,\}$$

The *MOP* Approach and *MOP* Solution (2)

Entry (N) and exit (X) information for basic block nodes must be pushed inside of the basic blocks:



The *MOP* Approach and *MOP* Solution (3)

...and their push to the instruction level:

Definition B.2.2.1 (The *MOP* Solution, Part 2)

The $MOP_{N,\iota}$ solution of $\mathcal{S}_G$ is defined by

$$MOP_{N,\iota}^{\mathcal{S}_G} : N \to (\mathcal{C}, \mathcal{C})$$

$$\forall\, n \in N.\ MOP_{N,\iota}^{\mathcal{S}_G}(n) =_{df} (\, N\text{-}MOP_{N,\iota}^{\mathcal{S}_G}(n),\ X\text{-}MOP_{N,\iota}^{\mathcal{S}_G}(n)\,)$$

The *MOP* Approach and *MOP* Solution (4)

where

$$N\text{-}MOP^{\mathcal{S}_G}_{N,\iota}(n) \;=_{df}\; \begin{cases} N\text{-}MOP^{\mathcal{S}_\mathbf{G}}_{\mathbf{N},\beta}(bb(n)) \\ \quad \text{if } n = start(bb(n)) \\ \\ [\![\, p \,]\!]_{N,\iota}(N\text{-}MOP^{\mathcal{S}_\mathbf{G}}_{\mathbf{N},\beta}(bb(n))) \\ \quad \text{otherwise } (p \text{ is the prefix path from} \\ \quad start(bb(n)) \text{ up to but exclusive of } n) \end{cases}$$

$$X\text{-}MOP^{\mathcal{S}_G}_{N,\iota}(n) \;=_{df}\; [\![\, p \,]\!]_{N,\iota}(N\text{-}MOP^{\mathcal{S}_\mathbf{G}}_{\mathbf{N},\beta}(bb(n)))$$

(p is the prefix path from $start(bb(n))$ up to and inclusive of n)

The *MaxFP* Approach and *MaxFP* Solution (1)

...for a node-labelled basic block flow graph.

Definition B.2.2.2 (The *MaxFP* Solution, Part 1)

The $MaxFP_{\mathbf{N},\beta}$ solution of $\mathcal{S}_{\mathbf{G}}$ is defined by

$$\forall\, \mathbf{n} \in \mathbf{N}.\ MaxFP^{\mathcal{S}_{\mathbf{G}}}_{\mathbf{N},\beta}(\mathbf{n}) =_{df} (\, N\text{-}MaxFP^{\mathcal{S}_{\mathbf{G}}}_{\mathbf{N},\beta}(\mathbf{n}),\ X\text{-}MaxFP^{\mathcal{S}_{\mathbf{G}}}_{\mathbf{N},\beta}(\mathbf{n})\,)$$

with

$$N\text{-}MaxFP^{\mathcal{S}_{\mathbf{G}}}_{\mathbf{N},\beta}(\mathbf{n}) =_{df} \nu\text{-}pre_{c_{\mathbf{s}},\beta}(\mathbf{n})$$

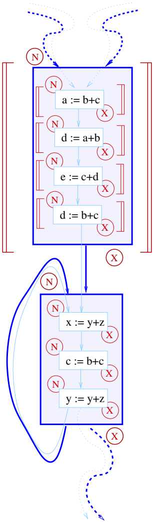
$$X\text{-}MaxFP^{\mathcal{S}_{\mathbf{G}}}_{\mathbf{N},\beta}(\mathbf{n}) =_{df} \nu\text{-}post_{c_{\mathbf{s}},\beta}(\mathbf{n})$$

where $\nu\text{-}pre_{c_{\mathbf{s}},\beta}$ and $\nu\text{-}post_{c_{\mathbf{s}},\beta}$ denote the greatest solution of the *MaxFP* Equation System for basic block graphs:

$$\begin{aligned}
\mathrm{pre}(\mathbf{n}) &= \begin{cases} c_{\mathbf{s}} & \text{if } \mathbf{n} = \mathbf{s} \\ \bigsqcap \{\, \mathrm{post}(\mathbf{m}) \mid \mathbf{m} \in pred_{\mathbf{G}}(\mathbf{n}) \,\} & \text{otherwise} \end{cases} \\
\mathrm{post}(\mathbf{n}) &= [\![\, \mathbf{n} \,]\!]_{\mathbf{N},\beta}(\mathrm{pre}(\mathbf{n}))
\end{aligned}$$

The *MaxFP* Approach and *MaxFP* Solution (2)

Entry (N) and exit (X) information for basic block nodes must be pushed inside of the basic blocks:



The *MaxFP* Approach and *MaxFP* Solution (3)

...and their push to the instruction level:

Definition B.2.2.2 (The *MaxFP* Solution, Part 2)

The $MaxFP_{N,\iota}$ solution of $\mathcal{S}_G$ is defined by

$$\forall\, n \in N.\ MaxFP^{\mathcal{S}_G}_{N,\iota}(n) =_{df} (\, N\text{-}MaxFP^{\mathcal{S}_G}_{N,\iota}(n),\ X\text{-}MaxFP^{\mathcal{S}_G}_{N,\iota}(n)\,)$$

with

$$N\text{-}MaxFP^{\mathcal{S}_G}_{N,\iota}(n) =_{df} \nu\text{-}pre_{c_s,\iota}(n)$$
$$X\text{-}MaxFP^{\mathcal{S}_G}_{N,\iota}(n) =_{df} \nu\text{-}post_{c_s,\iota}(n)$$

The *MaxFP* Approach and *MaxFP* Solution (4)

...where $\nu\text{-}pre_{c_s,\iota}$ and $\nu\text{-}post_{c_s,\iota}$ denote the greatest solution of the *MaxFP* Equation System for instruction graphs:

$$\text{pre}(n) = \begin{cases} \nu\text{-}pre_{c_s,\beta}(bb(n)) & \text{if } n = start(bb(n)) \\ \text{post}(m) & \text{otherwise, where } m \text{ is the unique predecessor of } n \text{ in } bb(n) \end{cases}$$

$$\text{post}(n) = \begin{cases} \nu\text{-}post_{c_s,\beta}(bb(n)) & \text{if } n = end(bb(n)) \\ [\![\, n \,]\!]_{N,\iota}(\text{pre}(n)) & \end{cases}$$

B.3

Available Expressions

B.3.1

Node-labelled Basic Block Graphs

Available Expressions (1)

...for node-labelled basic block graphs and a single term t.

Stage I: The Basic Block Level

Local Predicates (associated with basic block nodes):

- BB-XComp$_\beta^t$: β contains a statement ι computing t, and neither ι nor a statement following ι in β modifies an operand of t.
- BB-Transp$_\beta^t$: β does not contain a statement which modifies an operand of t.

The Basic Block *MaxFP* Equation System of Stage I:

$$\text{BB-N-Avail}_\beta = \begin{cases} c_{\mathbf{s}} & \text{if } \beta = \mathbf{s_G} \\ \bigwedge\limits_{\hat\beta \in pred(\beta)} \text{BB-X-Avail}_{\hat\beta} & \text{otherwise} \end{cases}$$

$$\text{BB-X-Avail}_\beta = (\text{BB-N-Avail}_\beta \wedge \text{BB-Transp}_\beta^t) \vee \text{BB-XComp}_\beta^t$$

Available Expressions (2)

Stage II: The Instruction Level

Local Predicates (associated with instruction nodes):

- Comp_ι^t: ι computes t.
- Transp_ι^t: ι does not modify an operand of t.
- ν-BB-N-Avail, ν-BB-X-Avail: the greatest solution of the *MaxFP* Equation System of Stage I.

Auxiliary Mappings

- *bb*: maps an instruction ι to the basic block β it is included in.
- *start*: maps a basic block β to its entry instruction ι.
- *end*: maps a basic block β to its exit instruction ι.

Available Expressions (3)

The Instruction *MaxFP* Equation System of Stage II:

$$\text{N-Avail}_\iota \quad = \quad \begin{cases} \nu\text{-BB-N-Avail}_{bb(\iota)} & \text{if } \iota = start(bb(\iota)) \\ \text{X-Avail}_{pred(\iota)} & \text{otherwise (note: } \mid pred(\iota) \mid = 1) \end{cases}$$

$$\text{X-Avail}_\iota \quad = \quad \begin{cases} \nu\text{-BB-X-Avail}_{bb(\iota)} & \text{if } \iota = end(bb(\iota)) \\ & \\ (\text{N-Avail}_\iota \vee \text{Comp}_\iota^t) \wedge \text{Transp}_\iota^t & \text{otherwise} \end{cases}$$

B.3.2

Node-labelled Instruction Graphs

Available Expressions

...for node-labelled instruction graphs and a single term t.

Local Predicates (associated with instruction nodes):

- Comp_ι^t: ι computes t.
- Transp_ι^t: ι does not modify an operand of t.

The Instruction *MaxFP* Equation System:

$$\text{N-Avail}_\iota = \begin{cases} c_{\mathbf{s}} & \text{if } \iota = \mathbf{s} \\ \bigwedge_{\hat{\iota} \in pred(\iota)} \text{X-Avail}_{\hat{\iota}} & \text{otherwise} \end{cases}$$

$$\text{X-Avail}_\iota = (\text{N-Avail}_\iota \vee \text{Comp}_\iota^t) \wedge \text{Transp}_\iota^t$$

B.3.3

Edge-labelled Instruction Graphs

Available Expressions

...edge-labelled instruction graphs and a single term t.

Locale Predicates (associated with instruction edges):

- $\mathrm{Comp}_{\varepsilon}^{t}$: Instruction ι of edge ε computes t.
- $\mathrm{Transp}_{\varepsilon}^{t}$: Instruction ι of edge ε does not modify an operand of t.

The Instruction *MaxFP* Equation System:

$$\mathrm{Avail}_n = \begin{cases} c_{\mathbf{s}} & \text{if } n = \mathbf{s} \\ \bigwedge\limits_{m \in pred(n)} (\mathrm{Avail}_m \vee \mathrm{Comp}^t_{(m,n)}) \wedge \mathrm{Transp}^t_{(m,n)} & \\ & \text{otherwise} \end{cases}$$

B.3.4

Summary of Findings

Findings

...edge-labelled instruction graphs are conceptually and notationally the most

- convenient ones.

...node-labelled basic block graphs the most

- inconvenient ones.

in the following

...we consider two more examples to illustrate the impact of selecting a flow graph variant on the conceptual and practical complexity of data flow analysis:

- Simple constants analysis (cf. Chap. B.4)
- Faint variables analysis (cf. Chap. B.5)

To this end we will oppose and investigate *MaxFP* formulations of these problems for

- node-labelled basic blcok graphs
- edge-labelled instruction graphs

which are the antipodes of each other.

B.4

Simple Constants

Simple Constants Analysis

...for the formal problem formulation we require two auxiliary functions:

- Backward substitution δ
- State transformation θ

together with their extensions to (path) instruction sequences.

Backward Substitution, State Transformation

Let $\iota \equiv (x := t)$ be an instruction. We define:

- Backward substitution δ_ι

  $\delta_\iota : \mathbf{T} \to \mathbf{T}$ defined by

  $$\forall\, s \in \mathbf{T}.\ \delta_\iota(s) =_{df} s[t/x]$$

  where $s[t/x]$ denotes the simultaneous replacement of all occurrences of x by t in s.

- State transformation θ_ι

  $\theta_\iota : \Sigma \to \Sigma$ defined by

  $$\forall\, \sigma \in \Sigma\ \forall\, v \in \mathbf{V}.\ \theta_\iota(\sigma)(v) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{if } v = x \\ \sigma(v) & \text{otherwise} \end{cases}$$

The Relationship of δ and θ

Let $\mathcal{I}$ denote the set of all instructions.

Lemma B.4.1 (Substitution Lemma for Instructions)

$$\forall\, \iota \in \mathcal{I}\ \forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma.\ \mathcal{E}(\delta_\iota(t))(\sigma) = \mathcal{E}(t)(\theta_\iota(\sigma))$$

Proof by induction on the structure of t.

B.4.1

Edge-labelled Instruction Graphs

Simple Constants Analysis

...for an edge-labelled instruction graph.

The *MaxFP* Equation System for edge-lab. instruction graphs:

$$\mathsf{SC}_n = \begin{cases} \sigma_{\mathbf{s}} & \text{falls } \; n = s \\ \lambda\, v.\ \prod \{\, \mathcal{E}(\delta_{(m,n)}(v))(\mathsf{SC}_m) \mid m \in pred(n) \,\} & \text{sonst} \end{cases}$$

where $\sigma_{\mathbf{s}} \in \Sigma$ start information.

The Solution of the Simple Constants Analysis is given by:

- ν-SC : $N \to \Sigma$, the greatest solution of the above EQS.

B.4.2

Node-labelled Basic Block Graphs

Backward Substitution, State Transformation

...for paths.

Adapting and extending δ and θ from instructions to sequences of instructions on paths (and hence basic blocks) of node-labelled flow graphs:

- Backward Substitution on Path Instruction Sequences

$$\Delta_p : \mathbf{T} \to \mathbf{T}$$

$$\Delta_p =_{df} \begin{cases} \delta_{n_q} & \text{if } q = 1 \\ \Delta_{(n_1,\ldots,n_{q-1})} \circ \delta_{n_q} & \text{if } q > 1 \end{cases}$$

- State Transformation on Path Instruction Sequences

$$\Theta_p : \Sigma \to \Sigma$$

$$\Theta_p =_{df} \begin{cases} \theta_{n_1} & \text{if } q = 1 \\ \Theta_{(n_2,\ldots,n_q)} \circ \theta_{n_1} & \text{if } q > 1 \end{cases}$$

The Relationship of Δ and Θ

Let $\mathcal{B}$ denote the set of all basic blocks.

Lemma B.4.2.1 (Substitution L. for Basic Blocks)

$$\forall\, \beta \in \mathcal{B}\ \forall\, t \in \mathbf{T}\ \forall\, \sigma \in \Sigma.\ \mathcal{E}(\Delta_\beta(t))(\sigma) = \mathcal{E}(t)(\Theta_\beta(\sigma))$$

Proof by induction on the length of β.

Simple Constants Analysis (1)

...for a node-labelled basic block graph.

Stage I: The Basic Block Level

The *BB_N MaxFP* Equation System of Stage I:

$$\text{BB-N-SC}_\beta \;=\; \begin{cases} \sigma_{\mathbf{s}} & \text{if } \beta = \mathbf{s} \\ \prod\{\text{BB-X-SC}_{\hat\beta} \mid \hat\beta \in \mathit{pred}(\beta)\} & \text{otherwise} \end{cases}$$

$$\text{BB-X-SC}_\beta \;=\; \lambda v.\ \mathcal{E}(\Delta_\beta(v))(\text{BB-N-SC}_\beta)$$

where $\sigma_{\mathbf{s}} \in \Sigma$ start information.

The Solution of the BB_N SC Analysis is given by:

- ν-BB-N-SC$_\beta$, ν-BB-X-SC$_\beta$: $\mathbf{N} \to \Sigma$, the greatest solutions of the above equation system.

Simple Constants Analysis (2)

Stage II: The Instruction Level

Auxiliary Mappings

- *bb*: maps an instruction ι to the basic block β it is included in.
- *start*: maps a basic block β to its entry instruction ι.
- *end*: maps a basic block β to its exit instruction ι.

The SI_N *MaxFP* Equation System of Stage II:

$$\text{N-SC}_\iota = \begin{cases} \nu\text{-BB-N-SC}_{bb(\iota)} & \text{if } \iota = start(bb(\iota)) \\ \text{X-SC}_{pred(\iota)} & \text{otherwise (because} \\ & \mid pred(\iota) \mid = 1) \end{cases}$$

$$\text{X-SC}_\iota = \begin{cases} \nu\text{-BB-X-SC}_{bb(\iota)} & \text{if } \iota = end(bb(\iota)) \\ \lambda v.\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) & \text{otherwise} \end{cases}$$

Simple Constants Analysis (3)

The Solution of the SI_N SC Analysis is given by:

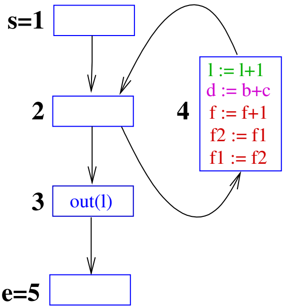
- ν-N-SC, ν-X-SC : $N \rightarrow \Sigma$, the greatest solution of the preceding equation system.

B.5

Faint Variables

Faint Variables: Between Life and Death

...consider the program:



Note: Instruction

- $l \ := \ l + 1$ is live,
- $d \ := \ b + c$ is dead,
- $f \ := \ f + 1$, $f1 \ := \ f2$, and $f2 \ := \ f1$ are live but faint (in German: geisterhaft, schattenhaft).

Faint Variables Analysis (1)

...for an edge-labelled instruction graphs.

Local Predicates (associated with instruction edges):

- $LifeEnforcingUse_\varepsilon^v$: Variable v is used by the instruction ι associated with edge ε and 'forced to live' by it (i.e., ι is an output or test operation).
- MOD_ε^v: The instruction ι at edge ε modifies variable v.
- $Ass\text{-}Used_\varepsilon^v$: Variable v, occurs in the right-hand side expression of the instruction ι associated with edge ε.

Auxiliary Mapping

- $LhsVar$: Maps an edge ε to the left-hand side variable of the instruction ι associated with it.

Faint Variables Analysis (2)

The *SI_E MaxFP* Equation System:

$$\mathrm{FAINT}^v_n \; = $$

$$\begin{cases} fv_{\mathbf{e}} & \text{if } n = \mathbf{e} \\ \displaystyle\bigwedge_{m \in succ(n)} \neg \mathrm{LifeEnforcingUse}^v_{(n,m)} \wedge \\ \qquad (\mathrm{FAINT}^v_m \;\vee\; \mathrm{MOD}^v_{(n,m)}) \wedge \\ \qquad (\mathrm{FAINT}^{LhsVar_{(n,m)}}_m \;\vee\; \neg\mathrm{Ass\text{-}Used}^v_{(n,m)}) & \text{otherwise} \end{cases}$$

where $fv_{\mathbf{e}} \in \mathbb{B}^{\mid \mathbf{V} \mid}$ start information.

The Solution of the *SI_E* Faint Variables Analysis is given by:

- ν-FAINT $: N \rightarrow \mathbb{B}^{\mid \mathbf{V} \mid}$, the greatest solution of the above equation system.

Informally

...a variable v ist faint at node n, if v

- is not forced to live by an instruction at an incoming edge of n (1-st conjunction term).
- is already faint at node n or modified by an instruction at an incoming edge of n and thereby made faint (2-nd conjunction term).
- is not used by an instruction at an incoming edge of n or (at most) used to assign a new value to a variable which is faint itself (3-rd conjunction term).

Summing up

Faint variables are an example of a (so-called) non-separable DFA problem, where a formulation leading to an efficient implementation is

- obvious for (node and edge-labelled) instruction graphs,
- not at all obvious, if not impossible at all, for (node and edge-labelled) basis block graphs.

  (Note that the naive straightforward extension to basic block graphs would require for every basic block **n** to compute the full semantic function $[\![\mathbf{n}]\!]_{faint} : \mathbb{B}^{k_\mathbf{n}} \to \mathbb{B}^{k_\mathbf{n}}$, where $k_\mathbf{n}$ is the number of variables occuring in **n**, a function with $2^{k_\mathbf{n}}$ arguments. In the worst case, $k_\mathbf{n}$ coincides even with the number of all variables in the program under consideration.)

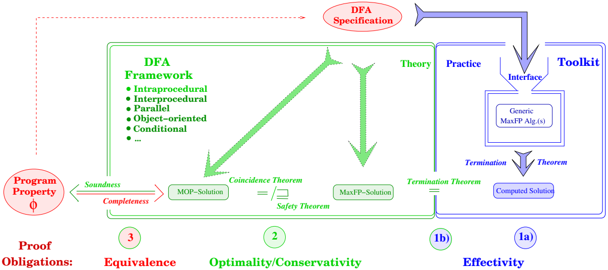
B.6

Conclusions

Conclusion

All 4 flow graph variants are conceptually essentially equivalent with in most cases only minor pragmatic advantages and disadvantages.

Thus the general holistic framework and tool kit view of DFA



is conceptually adequate and sufficient when being aware of the differences and their impact on specification, implementation, and proof obligation accomplishment.

B.7

References, Further Reading

Further Reading for Appendix B

- Larry Carter, Jeanne Ferrante, Clark Thomborson. *Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger*. In Conference Record of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2003), 106-114, 2003.
- Jens Knoop. *From DFA-Frameworks to DFA-Generators: A Unifying Multiparadigm Approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.
- Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65 - 79, 1998.