

Data flow analysis

Thomas Jensen

SOS
Master Science Informatique
Université Rennes

(Slides by Thomas Jensen and David Pichardie)

Outline

- 1 General introduction
- 2 Example of dataflow analysis : reachable definitions
- 3 Lattice theory
- 4 Resolution technique
- 5 Control flow graph and flow equations

Program analysis

Goal : deduce mechanically properties about a program's behaviour without executing it.

Application area : compilers, code optimisation, program verification, debugging...

3 rules :

- 1 The analyser must terminate.
- 2 The computed information must be correct.
- 3 It is allowed to return an approximative description of the program behaviour.

Static vs. dynamic

Static analysis :

- ▶ Work done at compile-time.
- ▶ Characterizes all executions.
- ▶ Conservative : approximates concrete program states.

Dynamic analysis :

- ▶ Run-time overhead.
- ▶ Characterizes one or a few executions.
- ▶ Precise : knows the concrete program state.
- ▶ Can't "look into the future".

Why abstraction?

The bad news : Rice's theorem :

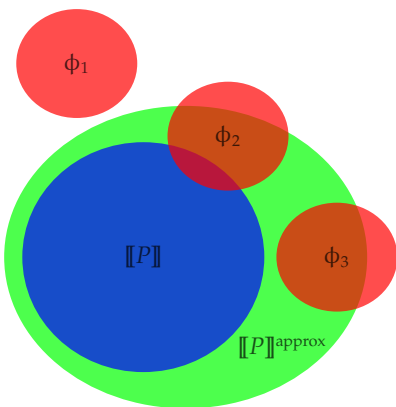
For a Turing-complete programming language, for any non-trivial property, the question of whether the computation of a given program satisfies this property is undecidable.

Solutions :

- ▶ verify the program interactively with the help of the user (deductive methods)
- ▶ computes only an **approximation** of the behavior of the program
 - ▶ Rice's theorem for static analyses :
No static analysis can decide a non-trivial property for all programs in a finite time.
 - ▶ It does not mean that it is impossible for *some* programs!
ASTRÉE¹ analyses the flight control codes of Airbus (~ 1 M loc)

1. <http://www.astree.ens.fr/>

A static analysis computes an approximation



- ▶ P is safe w.r.t. ϕ_1 and the analyser proves it

$$\llbracket P \rrbracket \cap \phi_1 = \emptyset \quad \llbracket P \rrbracket^{\text{approx}} \cap \phi_1 = \emptyset$$

- ▶ P is unsafe w.r.t. ϕ_2 and the analyser warns about it

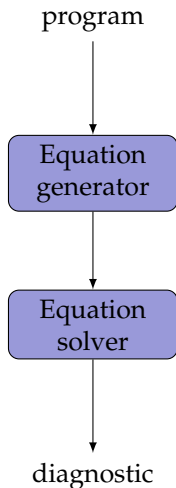
$$\llbracket P \rrbracket \cap \phi_2 \neq \emptyset \quad \llbracket P \rrbracket^{\text{approx}} \cap \phi_2 \neq \emptyset$$

- ▶ **but** P is safe w.r.t. ϕ_3 and the analyser can't prove it (this is called a *false alarm*)

$$\llbracket P \rrbracket \cap \phi_3 = \emptyset \quad \llbracket P \rrbracket^{\text{approx}} \cap \phi_3 \neq \emptyset$$

$\llbracket P \rrbracket$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1, ϕ_2, ϕ_3 :	erroneous/dangerous set of states	(computable)
$\llbracket P \rrbracket^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

Common structure of analyses



An analysis can be separated into two parts :

- 1 From a program description, producing an equation system (analysis specification)
 - ▶ the solutions of the system must be proved correct w.r.t. the program semantics
- 2 Solving the system
 - ▶ *fixpoint* iterations in *lattice* structures

This course

Several styles of program analysis will be presented in this course :

- ▶ Dataflow analysis.
- ▶ Constraint-based static analysis.
- ▶ Abstract Interpretation.

Dataflow analysis

- ▶ born in the 60s-70s (first optimising compilers)
- ▶ mainly target code optimisation (intermediate code)

Constraint-based analysis

- ▶ elegant technique for analysis specification/resolution

Abstract Interpretation

- ▶ a theory that unifies a large number of analyses
- ▶ semantic foundations to build correct analyses

Outline

- 1 General introduction
- 2 Example of dataflow analysis : reachable definitions
- 3 Lattice theory
- 4 Resolution technique
- 5 Control flow graph and flow equations

Dataflow analysis : examples

Reachable definitions : May a definition reach a given program point?
(Dependency analysis between instructions)

Available expressions : What are the expressions already computed at a given point?
(Re-use of computations)

Live variables : Is a variable used in the future?
(Assignment deletion, Register allocation)

Reachable definition analysis

Determine the set of definitions (assignments) that **may** reach a program point.

Factorial function :

```
1. y := x;  
2. z := 1;  
3. while y > 1 do  
4.   z := z * y;  
5.   y := y - 1;  
   end  
6. y := 0;
```

The variables used at point 4 may have been defined by the instructions at labels 1, 2, 4 and 5 (but not at label 6).

Reachable definition analysis

A **definition** is represented by a pair $(v, l) \in \text{Var} \times \text{Lab}^?$ with $\text{Lab}^? = \text{Lab} \cup \{?\}$.

(v, l) : “the variable v has been defined at program point l
and has not been modified since”

$(v, ?)$: “the variable v is not initialised”

We compute two sets at each label (program point) l :

$RD_{\text{in}}(l)$ = the definitions that enter l (*i.e.* reachable)

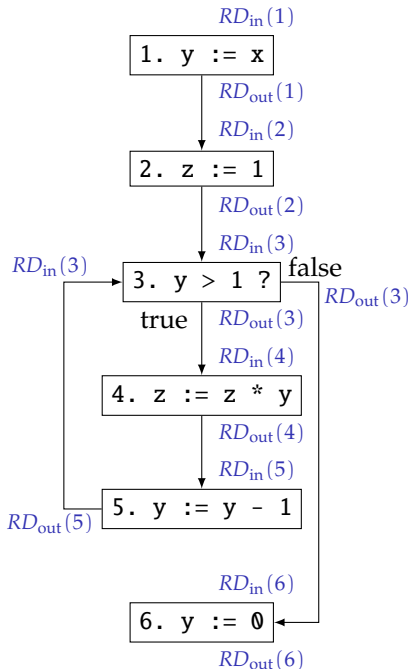
$RD_{\text{out}}(l)$ = the definitions that exit from l (auxiliary set)

Each instruction defines a relation between these sets.

```

1. y := x;
2. z := 1;
3. while y > 1 do
4.     z := z * y;
5.     y := y - 1;
6. end
7. y := 0;

```



Reachable definition analysis : equations (1)

An assignment deletes all previous definitions of the assigned variable.

$$RD_{out}(1) = RD_{in}(1) \setminus \{(y, l) \mid l \in Lab^2\} \cup \{(y, 1)\}$$

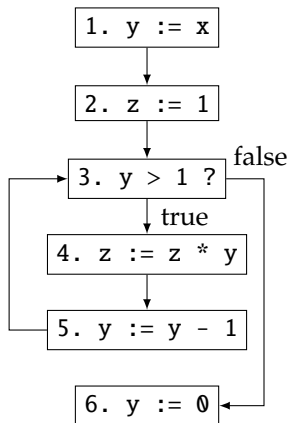
$$RD_{out}(2) = RD_{in}(2) \setminus \{(z, l) \mid l \in Lab^2\} \cup \{(z, 2)\}$$

$$RD_{out}(3) = RD_{in}(3)$$

$$RD_{out}(4) = RD_{in}(4) \setminus \{(z, l) \mid l \in Lab^2\} \cup \{(z, 4)\}$$

$$RD_{out}(5) = RD_{in}(5) \setminus \{(y, l) \mid l \in Lab^2\} \cup \{(y, 5)\}$$

$$RD_{out}(6) = RD_{in}(6) \setminus \{(y, l) \mid l \in Lab^2\} \cup \{(y, 6)\}$$



Reachable definition analysis : equations (2)

Definitions that are reachable after an instruction, are reachable before the next instruction.

$$RD_{in}(1) = \{(v, ?) \mid v \in Var\}$$

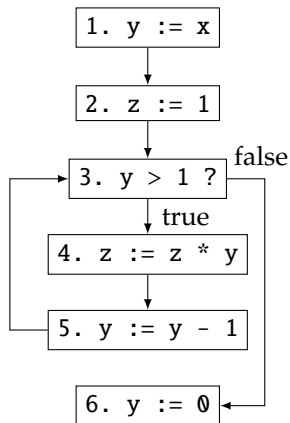
$$RD_{in}(2) = RD_{out}(1)$$

$$RD_{in}(3) = RD_{out}(2) \cup RD_{out}(5)$$

$$RD_{in}(4) = RD_{out}(3)$$

$$RD_{in}(5) = RD_{out}(4)$$

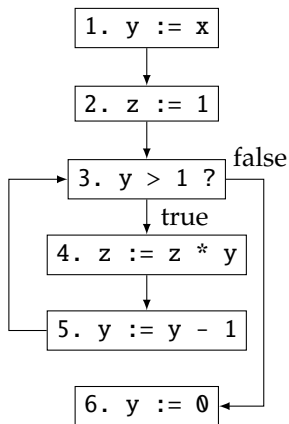
$$RD_{in}(6) = RD_{out}(3)$$



Reachable definition analysis : a solution

$$\begin{aligned}
 RD_{in}(1) &= \{(x, ?), (y, ?), (z, ?)\} \\
 RD_{in}(2) &= \{(x, ?), (y, 1), (z, ?)\} \\
 RD_{in}(3) &= \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
 RD_{in}(4) &= \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
 RD_{in}(5) &= \{(x, ?), (y, 1), (y, 5), (z, 4)\} \\
 RD_{in}(6) &= \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
 RD_{out}(1) &= \{(x, ?), (y, 1), (z, ?)\} \\
 RD_{out}(2) &= \{(x, ?), (y, 1), (z, 2)\} \\
 RD_{out}(3) &= \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
 RD_{out}(4) &= \{(x, ?), (y, 1), (y, 5), (z, 4)\} \\
 RD_{out}(5) &= \{(x, ?), (y, 5), (z, 4)\} \\
 RD_{out}(6) &= \{(x, ?), (y, 6), (z, 2), (z, 4)\}
 \end{aligned}$$

We observe that $(y, 1), (y, 5) \in RD_{in}(6)$.



Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{in}(l)$ and $RD_{out}(l)$ are initialised with \emptyset and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$\begin{array}{ll}
 RD_{in}(1) = \{(v, ?) \mid v \in Var\} & (e_1) \quad RD_{out}(1) = RD_{in}(1) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 1)\} (s_1) \\
 RD_{in}(2) = RD_{out}(1) & (e_2) \quad RD_{out}(2) = RD_{in}(2) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 2)\} (s_2) \\
 RD_{in}(3) = RD_{out}(2) \cup RD_{out}(5) & (e_3) \quad RD_{out}(3) = RD_{in}(3) (s_3) \\
 RD_{in}(4) = RD_{out}(3) & (e_4) \quad RD_{out}(4) = RD_{in}(4) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 4)\} (s_4) \\
 RD_{in}(5) = RD_{out}(4) & (e_5) \quad RD_{out}(5) = RD_{in}(5) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 5)\} (s_5) \\
 RD_{in}(6) = RD_{out}(3) & (e_6) \quad RD_{out}(6) = RD_{in}(6) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 6)\} (s_6)
 \end{array}$$

Iteration

$$\begin{array}{ll}
 RD_{in}(1) = & RD_{out}(1) = \\
 RD_{in}(2) = & RD_{out}(2) = \\
 RD_{in}(3) = & RD_{out}(3) = \\
 RD_{in}(4) = & RD_{out}(4) = \\
 RD_{in}(5) = & RD_{out}(5) = \\
 RD_{in}(6) = & RD_{out}(6) =
 \end{array}$$

Stabilisation!

Reachable definition analysis : several solutions ?

The equation system admits **several solutions!**

Equations :

$$\begin{array}{ll}
 RD_{in}(1) = \{(v, ?) \mid v \in Var\} & RD_{out}(1) = RD_{in}(1) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 1)\} \\
 RD_{in}(2) = RD_{out}(1) & RD_{out}(2) = RD_{in}(2) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 2)\} \\
 RD_{in}(3) = RD_{out}(2) \cup RD_{out}(5) & RD_{out}(3) = RD_{in}(3) \\
 RD_{in}(4) = RD_{out}(3) & RD_{out}(4) = RD_{in}(4) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 4)\} \\
 RD_{in}(5) = RD_{out}(4) & RD_{out}(5) = RD_{in}(5) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 5)\} \\
 RD_{in}(6) = RD_{out}(3) & RD_{out}(6) = RD_{in}(6) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 6)\}
 \end{array}$$

Previous solution : Another solution :

$$\begin{array}{ll}
 RD'_{in}(1) = \{(x, ?), (y, ?), (z, ?)\} & RD'_{out}(1) = \{(x, ?), (y, 1), (z, ?)\} \\
 RD'_{in}(2) = \{(x, ?), (y, 1), (z, ?)\} & RD'_{out}(2) = \{(x, ?), (y, 1), (z, 2)\} \\
 RD'_{in}(3) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 2), (z, 4)\} & RD'_{out}(3) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
 RD'_{in}(4) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 2), (z, 4)\} & RD'_{out}(4) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 4)\} \\
 RD'_{in}(5) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 4)\} & RD'_{out}(5) = \{(x, ?), (x, 1), (y, 5), (z, 4)\} \\
 RD'_{in}(6) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 2), (z, 4)\} & RD'_{out}(6) = \{(x, ?), (x, 1), (y, 6), (z, 2), (z, 4)\}
 \end{array}$$

Choosing the best solution

Remark :

$$RD_{in}(1) \subseteq RD'_{in}(1), \quad RD_{out}(1) \subseteq RD'_{out}(1), \quad \dots, \quad RD_{out}(6) \subseteq RD'_{out}(6)$$

RD gives more precise information than RD' :

▶ $RD_{in}(3) = \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\}$

“the value of x at point 3 is not initialised”

▶ $RD'_{in}(3) = \{(x, ?), (x, 1), (y, 1), (y, 5), (z, 2), (z, 4)\}$

“the value of x at point 3 is not initialised, or has been defined at point 1”

Between two comparable solutions (e.g. $\vec{RD} \subseteq \vec{RD}'$), we prefer the smallest.

Theoretical result : there always exists a smallest solution

Equation resolution

The previous analysis is a solution of an equation system of the form

$$\begin{cases} x_1 &= f_1(x_1, \dots, x_n) \\ &\vdots \\ x_n &= f_n(x_1, \dots, x_n) \end{cases} \quad \text{or} \quad \vec{x} = \vec{f}(\vec{x})$$

called **fixpoint equations**.

It is a common mathematical problem that raises two questions :

- ① Existence and uniqueness (in what sense?) of the solution?
- ② Effective computation method?

A few observations about the previous analysis :

- ▶ The x_i are **sets** that can be ordered by set inclusion \subseteq
 - ▶ The functions f_i are **monotone** (increasing) for the partial order \subseteq
- \Rightarrow Suitable mathematical framework : **lattice theory**

Outline

- 1 General introduction
- 2 Example of dataflow analysis : reachable definitions
- 3 Lattice theory
- 4 Resolution technique
- 5 Control flow graph and flow equations

Poset

Definition

A *partially ordered set (poset)* is a couple (A, \sqsubseteq) with A a set, and \sqsubseteq a partial order relation, i.e. :

$$\forall x \in A, x \sqsubseteq x \quad (\text{reflexivity})$$

$$\forall x, y \in A, x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y \quad (\text{antisymmetry})$$

$$\forall x, y, z \in A, x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z \quad (\text{transitivity})$$

Examples

- ▶ (\mathbb{N}, \leq) (total : $\forall x, y, x \leq y \vee y \leq x$)
- ▶ $(\mathbb{N}, \text{"is a divisor of"})$ written $(\mathbb{N}, |)$
- ▶ $(\mathcal{P}(X), \subseteq)$ with X any set
- ▶ $(A^*, \text{"to be a prefix of"})$ with A an alphabet

Exercise

Show that $(\mathbb{N}, \text{"is a divisor of"})$ is a poset.

Hasse diagram

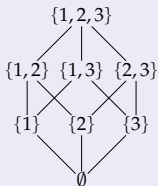
Graphical representation of a poset

Definition

A 2D-drawing (set of points and segments) is a *Hasse diagram* of a poset (A, \sqsubseteq) iff

- ▶ each element of A is associated with a point
- ▶ if $x \sqsubseteq y$ with $x \neq y$ and $\neg \exists z, x \sqsubseteq z \sqsubseteq y$ then
 - ▶ a segment connects the points p_x and p_y that are associated respectively with x and y
 - ▶ the ordinate (vertical scale) of p_x is lower than the ordinate of p_y

Example



is an Hasse diagram of the poset $(\mathcal{P}(\{1,2,3\}), \subseteq)$

Exercise

Give a Hasse diagram of the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$

Lattice

Definition

A *lattice* is a 4-tuple $(A, \sqsubseteq, \sqcup, \sqcap)$ with

- ▶ (A, \sqsubseteq) a poset,
- ▶ \sqcup a binary least upper bound :

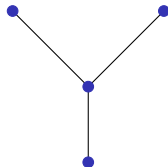
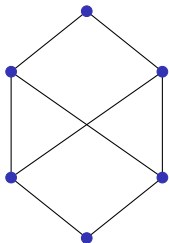
$$\begin{aligned} \forall x, y \in A, x \sqsubseteq x \sqcup y \wedge y \sqsubseteq x \sqcup y \\ \forall x, y, z \in A, x \sqsubseteq z \wedge y \sqsubseteq z \Rightarrow x \sqcup y \sqsubseteq z \end{aligned}$$

- ▶ \sqcap a binary greatest lower bound :

$$\begin{aligned} \forall x, y \in A, x \sqcap y \sqsubseteq x \wedge x \sqcap y \sqsubseteq y \\ \forall x, y, z \in A, z \sqsubseteq x \wedge z \sqsubseteq y \Rightarrow z \sqsubseteq x \sqcap y \end{aligned}$$

Exercise

Between the following diagrams, which represent lattices?



Exercise

Give the lattice structure of (\mathbb{N}, \leq) and $(\mathbb{N}, |)$.

Complete lattice

Definition

A complete lattice is a triple $(A, \sqsubseteq, \bigsqcup)$ with

- ▶ (A, \sqsubseteq) a poset,
- ▶ \bigsqcup a least upper bound : for all subsets S of A ,
 - ▶ $\forall a \in S, a \sqsubseteq \bigsqcup S$
 - ▶ $\forall b \in A, (\forall a \in S, a \sqsubseteq b) \Rightarrow \bigsqcup S \sqsubseteq b$

A complete lattice necessarily possesses a *greatest lower bound* \sqcap operator, *i.e.* : for all subsets S of A ,

- ▶ $\forall a \in S, \sqcap S \sqsubseteq a$
- ▶ $\forall b \in A, (\forall a \in S, b \sqsubseteq a) \Rightarrow b \sqsubseteq \sqcap S$

Just consider

$$\sqcap S = \bigsqcup \{ y \mid \forall x \in S, y \sqsubseteq x \}.$$

Examples

- 1 For all set X , $(\mathcal{P}(X), \subseteq, \cup)$ is a complete lattice for which \bigcap is a greatest lower bound.
- 2 Every finite lattice is complete.

Exercise

Show that any complete lattice admits

- ▶ a greatest element \top ($\forall x, x \sqsubseteq \top$)
- ▶ a least element \perp ($\forall x, \perp \sqsubseteq x$)

Fixpoints, post-fixpoints and pre-fixpoints

Definition

Consider $f \in A \rightarrow A$ with (A, \sqsubseteq) a poset, an element $x \in A$

- ▶ is a *fixpoint* of f iff $f(x) = x$
- ▶ is a *greatest fixpoint* of f iff $f(x) = x$ and $\forall y, f(y) = y \Rightarrow y \sqsubseteq x$
- ▶ is a *least fixpoint* of f iff $f(x) = x$ and $\forall y, f(y) = y \Rightarrow x \sqsubseteq y$
- ▶ is a *post-fixpoint* of f iff $f(x) \sqsubseteq x$
- ▶ is a *pre-fixpoint* of f iff $x \sqsubseteq f(x)$

Definition

Let $f \in A \rightarrow A$, f is *monotone* iff

$$\forall x, y \in A, x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

Fixpoints, post-fixpoints and pre-fixpoints

Theorem (Knaster-Tarski)

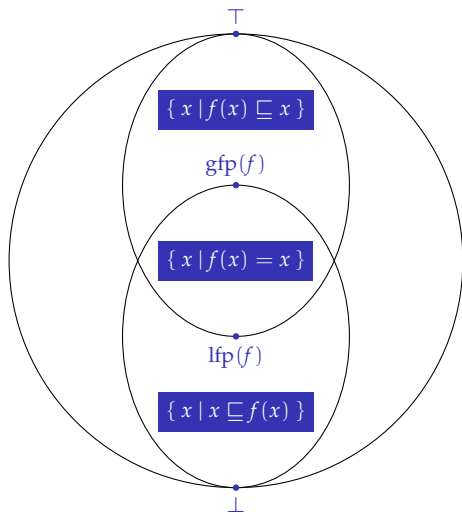
In a complete lattice (A, \sqsubseteq, \sqcap) , for all monotone functions $f \in A \rightarrow A$,

- ▶ *the least fixpoint $\text{lfp}(f)$ of f exists and is $\sqcap \{x \in A \mid f(x) \sqsubseteq x\}$,*
- ▶ *the greatest fixpoint $\text{gfp}(f)$ of f exists and is $\sqcup \{x \in A \mid x \sqsubseteq f(x)\}$,*

Proof of the Knaster-Tarski theorem

- ▶ Let us define $a = \bigsqcap P$ with $P = \text{PostFix}(f) = \{x \mid f(x) \sqsubseteq x\}$.
- ▶ For all $x \in P$, we have
 - $a \sqsubseteq x$ a greatest lower bound of P
 - $f(a) \sqsubseteq f(x)$ f monotone
 - $f(a) \sqsubseteq x$ def. P and transitivity
 hence $f(a) \sqsubseteq x$ for all $x \in P$
- ▶ $f(a) \sqsubseteq a$ previous result and a greatest lower bound of P
 $\Rightarrow f(f(a)) \sqsubseteq f(a)$ f monotone
 $\Rightarrow f(a) \in P$ def. P
 $\Rightarrow a \sqsubseteq f(a)$ a lower bound of P
 $\Rightarrow f(a) = a$ antisymmetry
 hence $a \in \text{Fix}(f)$
- ▶ If $x \in \text{Fix}(f)$ then $x \in P$, hence $a \sqsubseteq x$ because a is a lower bound of P .
 Hence $a = \text{lfp}(f)$.
- ▶ Proof of $\text{gfp}(f) = \bigsqcup \text{PreFix}(f)$ by duality.

Fixpoints, post-fixpoints and pre-fixpoints



$$\begin{aligned} \top &= \bigsqcup \{x \mid f(x) \sqsubseteq x\} \\ \text{gfp}(f) &= \bigsqcup \{x \mid x \sqsubseteq f(x)\} \\ \text{lfp}(f) &= \bigsqcap \{x \mid f(x) \sqsubseteq x\} \\ \perp &= \bigsqcap \{x \mid x \sqsubseteq f(x)\} \end{aligned}$$

Fixpoint computation

Theorem

Let (A, \sqsubseteq) be a poset with a least element \perp .
 Let f be a monotone function. If the sequence

$$\perp, f(\perp), \dots, f^n(\perp), \dots$$

stabilises from some k (i.e. $f^k(\perp) = f^{k+1}(\perp)$), then $f^k(\perp)$ is the least fixpoint of f .

Proof : Since $\perp \sqsubseteq f(\perp)$ and f is monotone, we can show by induction on \mathbb{N} that $\perp, f(\perp), \dots, f^n(\perp), \dots$ is an increasing sequence.

Let k such that $f^k(\perp) = f^{k+1}(\perp)$.

- ▶ Hence $f^k(\perp)$ is a fixpoint of f .
- ▶ If x is a fixpoint of f , we show by induction on \mathbb{N} that $f^n(\perp) \sqsubseteq x \forall n \in \mathbb{N}$.
 It shows in particular that $f^k(\perp) \sqsubseteq x$.

Remark : $\top, f(\top), \dots, f^n(\top), \dots$ allows to compute the greatest fixpoint of f .

Fixpoint computation : ascending chain condition

Definition

A poset (A, \sqsubseteq) verifies the ascending chain condition if for all ascending (increasing) sequence $x_0 \sqsubseteq x_1 \sqsubseteq \dots \sqsubseteq x_n \sqsubseteq \dots$ there exists an index k from which the sequence is stationary ($\forall n \geq k, x_k = x_n$) (i.e. the sequence eventually stabilises).

Corollary

Let (A, \sqsubseteq) a poset that verifies the ascending chain condition and f a monotone function. The sequence $\perp, f(\perp), \dots, f^n(\perp), \dots$ eventually stabilises. Its limit is the least fixpoint of f .

Remark : A finite poset verifies the ascending chain condition.

Fixpoint computation : Kleene fixpoint theorem

Definition

Let (A, \sqsubseteq, \sqcup) a complete lattice. A function $f \in A \rightarrow A$ is **continuous** iff

$$\forall S \subseteq A, \sqcup f(S) = f(\sqcup S)$$

Remark : a continuous function is necessarily monotone.

Theorem (Kleene fixpoint theorem)

In a complete lattice (A, \sqsubseteq, \sqcup) , for all continuous function $f \in A \rightarrow A$, the least fixpoint $\mathbf{lfp}(f)$ of f is equal to $\sqcup \{f^n(\perp) \mid n \in \mathbb{N}\}$.

Remark : the original theorem is stated for *complete partial order* (CPO).

Proof of the Kleene fixpoint theorem

► We have already shown that $f^n(\perp) \sqsubseteq f^{n+1}(\perp)$

► $\bigsqcup_{n \geq 0} f^n(\perp)$ is a fixpoint of f :

$$\begin{aligned}
 & f\left(\bigsqcup_{n \geq 0} f^n(\perp)\right) \\
 &= \bigsqcup_{n \geq 0} f(f^n(\perp)) && f \text{ is continuous} \\
 &= f^0(\perp) \sqcup \bigsqcup_{n \geq 0} f^{n+1}(\perp) && (\perp \sqcup x = x) \text{ and def. } f^n \\
 &= \bigsqcup_{n \geq 0} f^n(\perp)
 \end{aligned}$$

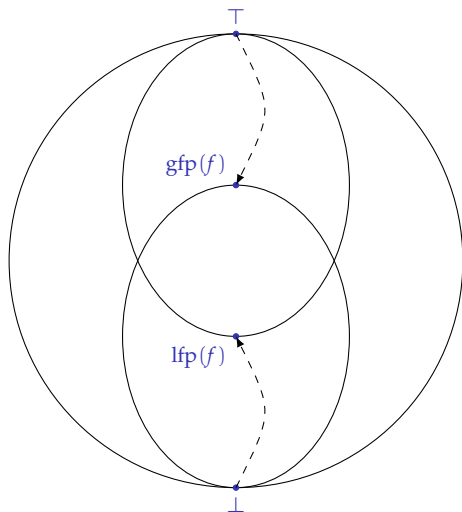
► It is the least fixpoint : consider $x \in \text{Fix}(f)$

$$- f^0(\perp) = \perp \sqsubseteq x$$

$$- \forall n \geq 0 : f^n(\perp) \sqsubseteq x \quad \text{induction on } n, \text{ because } f \text{ monotone and } f(x) = x$$

$$- \bigsqcup_{n \geq 0} f^n(\perp) \sqsubseteq x \quad \text{greater bound}$$

Fixpoint computation



$$\top, f(\top), \dots, f^n(\top), \dots, \text{gfp } f$$

$$\perp, f(\perp), \dots, f^n(\perp), \dots, \text{lfp } f$$

Outline

- 1 General introduction
- 2 Example of dataflow analysis : reachable definitions
- 3 Lattice theory
- 4 Resolution technique
- 5 Control flow graph and flow equations

The underlying lattice structure of the Reaching definitions analysis

$(\mathcal{P}(Var \times Lab^?), \subseteq, \cup)$ is a complete lattice.

Lattice product : if $(L_1, \subseteq_1, \sqcup_1)$ and $(L_2, \subseteq_2, \sqcup_2)$ are complete lattices, their product $L_1 \times L_2$ is the complete lattice $(L_1 \times L_2, \subseteq_{L_1 \times L_2}, \sqcup_{L_1 \times L_2})$ defined by :

$$\begin{aligned} (x_1, x_2) \subseteq_{L_1 \times L_2} (y_1, y_2) &\Leftrightarrow x_1 \subseteq_1 y_1 \wedge x_2 \subseteq_2 y_2 \\ \sqcup_{L_1 \times L_2} S &= (\sqcup_1 proj_1(S), \sqcup_2 proj_2(S)), \forall S \subseteq L_1 \times L_2 \end{aligned}$$

Conclusion :

$$(RD_s(1), RD_e(1), \dots, RD_s(6), RD_e(6)) \in \mathcal{P}(Var \times Lab^?)^{12}$$

and $(\mathcal{P}(Var \times Lab^?)^{12}, \subseteq^{12}, \cup^{12})$ is a complete lattice.

Exercise : Justify the termination of the analysis.

Accelerated iterations

Consider the system

$$\begin{cases} x_1 &= f_1(x_1, \dots, x_n) \\ \vdots & \\ x_n &= f_n(x_1, \dots, x_n) \end{cases}$$

Standard iterations :

$$\begin{aligned} x_1^{i+1} &= f_1(x_1^i, \dots, x_n^i) \\ x_2^{i+1} &= f_2(x_1^i, \dots, x_n^i) \\ &\vdots \\ x_n^{i+1} &= f_n(x_1^i, \dots, x_n^i) \end{aligned}$$

Chaotic iterations : at each step, only iterate selected equations, without forgetting any equation infinitely often.

$L \in \mathbb{N} \rightarrow \mathcal{P}(\{1, \dots, n\})$ gives the iteration strategy (*i.e.* at the i^{th} iteration, equations in L_i are used).

$$\begin{aligned} x_j^{i+1} &= f_j(x_1^i, \dots, x_n^i) && \text{if } j \in L_{i+1} \\ x_j^{i+1} &= x_j^i && \text{if } j \notin L_{i+1} \end{aligned}$$

Faster propagation and convergence (fewer unnecessary computations)
 A good iteration order depends on the dependences between the x_i .

Example

Chaotic iteration on the “reaching definitions” example.

First : the equation system can be simplified (at least by hand).

$$\begin{array}{ll}
 RD_{in}(1) = \{(v, ?) \mid v \in Var\} & (e_1) \quad RD_{out}(1) = RD_{in}(1) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 1)\} (s_1) \\
 RD_{in}(2) = RD_{out}(1) & (e_2) \quad RD_{out}(2) = RD_{in}(2) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 2)\} (s_2) \\
 RD_{in}(3) = RD_{out}(2) \cup RD_{out}(5) & (e_3) \quad RD_{out}(3) = RD_{in}(3) (s_3) \\
 RD_{in}(4) = RD_{out}(3) & (e_4) \quad RD_{out}(4) = RD_{in}(4) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 4)\} (s_4) \\
 RD_{in}(5) = RD_{out}(4) & (e_5) \quad RD_{out}(5) = RD_{in}(5) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 5)\} (s_5) \\
 RD_{in}(6) = RD_{out}(3) & (e_6) \quad RD_{out}(6) = RD_{in}(6) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 6)\} (s_6)
 \end{array}$$

Example

It is hence sufficient to solve the following system :

$$RD_{\text{out}}(3) = \{(x, ?), (y, 1), (z, 2)\} \cup RD_{\text{out}}(5) \quad (s_3)$$

$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 4)\} \quad (s_4)$$

$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 5)\} \quad (s_5)$$

$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 6)\} \quad (s_6)$$

L_i		$\{s_3\}$	$\{s_4\}$	$\{s_5\}$
$RD_{\text{out}}(3)$	\emptyset	$\{(x, ?), (y, 1), (z, 2)\}$	$\{(x, ?), (y, 1), (z, 2)\}$	$\{(x, ?), (y, 1), (z, 2)\}$
$RD_{\text{out}}(4)$	\emptyset	\emptyset	$\{(x, ?), (y, 1), (z, 4)\}$	$\{(x, ?), (y, 1), (z, 4)\}$
$RD_{\text{out}}(5)$	\emptyset	\emptyset	\emptyset	$\{(x, ?), (y, 5), (z, 4)\}$
$RD_{\text{out}}(6)$	\emptyset	\emptyset	\emptyset	\emptyset

$\{s_3\}$	$\{s_4, s_6\}$
$\{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\}$	$\{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\}$
$\{(x, ?), (y, 1), (z, 4)\}$	$\{(x, ?), (y, 1), (y, 5), (z, 4)\}$
$\{(x, ?), (y, 5), (z, 4)\}$	$\{(x, ?), (y, 5), (z, 4)\}$
\emptyset	$\{(x, ?), (y, 6), (z, 2), (z, 4)\}$

The Work-set Algorithm

```

forall  $i \in \{1, \dots, n\}$  do  $x_i := \perp$ ;
 $W := \{1, \dots, n\}$ 
repeat
   $i := \text{choose}(W)$ ;
   $tmp := f_i(x_1, \dots, x_n)$ ;
  if  $tmp \neq x_i$  then begin  /* the value of  $x_i$  has changed */
     $x_i := tmp$ ;
     $W := W \cup \text{dependencies}(x_i)$ 
  until  $W = \emptyset$ 

```

$\text{choose}(S)$: removes one element from a set S .

$\text{dependencies}(x)$: returns the set of variables that depends on a variable x .

The chaotic iteration (*i.e.* the choice of a good iteration order) can be combined with the “work-set” technique (re-computation only when necessary).

Outline

- 1 General introduction
- 2 Example of dataflow analysis : reachable definitions
- 3 Lattice theory
- 4 Resolution technique
- 5 Control flow graph and flow equations

Control flow graph

- ▶ Definition of the **While** language with labels. Labels designate the **program points** where we stock some data flow information.
- ▶ Control flow graph of program and extraction from labeled programs.
- ▶ Generating data flow equations from control flow graphs.

The While language

Syntax

- ▶ Integers ($n \in \mathit{Num}$)
- ▶ Variables ($x \in \mathit{Var}$)
- ▶ Arithmetic expressions ($a \in \mathit{Aexp}$)

$$a ::= n \mid x \mid a_1 + a_2 \mid \dots$$

- ▶ Boolean expressions ($b \in \mathit{Bexp}$)

$$b ::= \mathit{true} \mid \mathit{false} \mid a_1 \otimes a_2 \mid \neg b \mid b_1 \mathit{and} b_2 \mid \dots$$

The While language with labels

We re-use the syntax of **While** that has been presented in the lecture on operational semantics.

$$S ::= [\mathbf{x} := a]^l \mid [\mathbf{skip}]^l \mid S_1; S_2 \mid \mathbf{if} [b]^l \mathbf{then} S_1 \mathbf{else} S_2 \mid \mathbf{while} [b]^l \mathbf{do} S$$

$$n \in \mathit{Num}, \mathbf{x} \in \mathit{Var}, a \in \mathit{Aexp}, b \in \mathit{Bexp}, S \in \mathit{Comm}, l \in \mathit{Lab}$$

enriched with labels.

The labels allow to attach the analysis results to program points.

Control flow graph

We associate with each instruction $S \in \text{Comm}$:

$\text{init}(S)$	$\in \text{Lab}$:	entry point label of S
$\text{final}(S)$	$\subseteq \text{Lab}$:	exit point labels of S
$\text{labels}(S)$	$\subseteq \text{Lab}$:	labels which appear in S
$\text{flow}(S)$	$\subseteq \text{Lab} \times \text{Lab}$:	edges of the control flow graph

Example :

$\text{power} = [\mathbf{z} := 1]^1; \text{while } [\mathbf{x} > 0]^2 \text{ do } ([\mathbf{z} := \mathbf{z} * \mathbf{y}]^3; [\mathbf{x} := \mathbf{x} - 1]^4)$

$\text{init}(\text{power})$	=	1
$\text{final}(\text{power})$	=	{2}
$\text{labels}(\text{power})$	=	{1, 2, 3, 4}
$\text{flow}(\text{power})$	=	{(1, 2), (2, 3), (3, 4), (4, 2)}

Control flow graph

Each function is defined by induction on the **While** syntax.

$$\begin{aligned}
 \mathit{init}([x := a]^l) &= l \\
 \mathit{init}([\mathbf{skip}]^l) &= l \\
 \mathit{init}(S_1; S_2) &= \mathit{init}(S_1) \\
 \mathit{init}(\mathbf{if} [b]^l \mathbf{then} S_1 \mathbf{else} S_2) &= l \\
 \mathit{init}(\mathbf{while} [b]^l \mathbf{do} S) &= l
 \end{aligned}$$

$$\begin{aligned}
 \mathit{final}([x := a]^l) &= \{l\} \\
 \mathit{final}([\mathbf{skip}]^l) &= \{l\} \\
 \mathit{final}(S_1; S_2) &= \mathit{final}(S_2) \\
 \mathit{final}(\mathbf{if} [b]^l \mathbf{then} S_1 \mathbf{else} S_2) &= \mathit{final}(S_1) \cup \mathit{final}(S_2) \\
 \mathit{final}(\mathbf{while} [b]^l \mathbf{do} S) &= \{l\}
 \end{aligned}$$

Control flow graph

$$\begin{aligned}
 \mathit{labels}([x := a]^l) &= \{l\} \\
 \mathit{labels}([\mathbf{skip}]^l) &= \{l\} \\
 \mathit{labels}(S_1; S_2) &= \mathit{labels}(S_1) \cup \mathit{labels}(S_2) \\
 \mathit{labels}(\mathbf{if} [b]^l \mathbf{then} S_1 \mathbf{else} S_2) &= \{l\} \cup \mathit{labels}(S_1) \cup \mathit{labels}(S_2) \\
 \mathit{labels}(\mathbf{while} [b]^l \mathbf{do} S) &= \{l\} \cup \mathit{labels}(S)
 \end{aligned}$$

Control flow graph

$$\begin{aligned}
 \mathit{flow}([x := a]^l) &= \emptyset \\
 \mathit{flow}([\mathbf{skip}]^l) &= \emptyset \\
 \mathit{flow}(S_1; S_2) &= \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
 &\quad \cup \{(l, \mathit{init}(S_2)) \mid l \in \mathit{final}(S_1)\} \\
 \mathit{flow}(\mathbf{if} [b]^l \mathbf{then} S_1 \mathbf{else} S_2) &= \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
 &\quad \cup \{(l, \mathit{init}(S_1)), (l, \mathit{init}(S_2))\} \\
 \mathit{flow}(\mathbf{while} [b]^l \mathbf{do} S) &= \mathit{flow}(S) \cup \{(l, \mathit{init}(S))\} \\
 &\quad \cup \{(l', l) \mid l' \in \mathit{final}(S)\}
 \end{aligned}$$

We only consider programs with distinct labels

- ▶ for all $l \in \mathit{labels}(S)$, $[B]^l \in S$ identifies, without ambiguity, the elementary block ($[x := a]^l$ or $[\mathbf{skip}]^l$) or the test ($[b]^l$) which appears in S .

We suppose also that no flow reaches the initial point of the program.

Remember : reachable definition analysis

$$RD_{in}(1) = \{(v, ?) \mid v \in Var\}$$

$$RD_{out}(1) = RD_{in}(1) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 1)\}$$

$$RD_{in}(2) = RD_{out}(1)$$

$$RD_{out}(2) = RD_{in}(2) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 2)\}$$

$$RD_{in}(3) = RD_{out}(2) \cup RD_{out}(5)$$

$$RD_{out}(3) = RD_{in}(3)$$

$$RD_{in}(4) = RD_{out}(3)$$

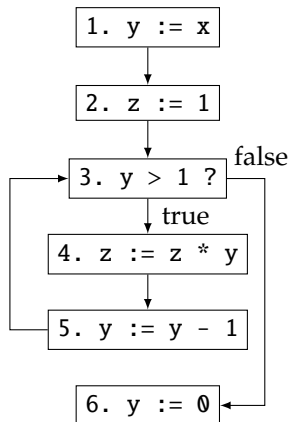
$$RD_{out}(4) = RD_{in}(4) \setminus \{(z, l) \mid l \in Lab^?\} \cup \{(z, 4)\}$$

$$RD_{in}(5) = RD_{out}(4)$$

$$RD_{out}(5) = RD_{in}(5) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 5)\}$$

$$RD_{in}(6) = RD_{out}(3)$$

$$RD_{out}(6) = RD_{in}(6) \setminus \{(y, l) \mid l \in Lab^?\} \cup \{(y, 6)\}$$



Data flow equations of reachable definitions

Domain of data flow properties :

$$RD_{in}(l), RD_{out}(l) \in \wp(\text{Var} \times \text{Lab}^?)$$

“Gen-kill” rules

$$kill([\mathbf{x} := a]^l) = \{(\mathbf{x}, l') \mid l' \in \text{Lab}^?\}$$

$$kill([\mathbf{skip}]^l) = \emptyset$$

$$kill([b]^l) = \emptyset$$

$$gen([\mathbf{x} := a]^l) = \{(x, l)\}$$

$$gen([\mathbf{skip}]^l) = \emptyset$$

$$gen([b]^l) = \emptyset$$

For all program points $[i]^l \in P$,

$$RD_{in}(l) = \begin{cases} \{(\mathbf{x}, ?) \mid \mathbf{x} \in \text{Var}\} & \text{if } l = \mathit{init}(P) \\ \bigcup_{(l', l) \in \mathit{flow}(P)} RD_{out}(l') & \end{cases}$$

$$RD_{out}(l) = RD_{in}(l) \setminus kill([i]^l) \cup gen([i]^l)$$

Available expressions

Determine the expressions whose value is already available in a variable.

Domain of values : $AE_{in}(l), AE_{out}(l) \in \mathcal{P}(Var \times Aexp)$

$$kill([x := a]^l) = \{(v, e) \mid x \in Var(e)\} \cup \{(x, e) \mid e \in Aexp\}$$

$$kill([\mathbf{skip}]^l) = \emptyset$$

$$kill([b]^l) = \emptyset$$

$$gen([x := a]^l) = \{(x, a) \mid x \notin Var(a)\}$$

$$gen([\mathbf{skip}]^l) = \emptyset$$

$$gen([b]^l) = \emptyset$$

For all program points $[i]^l \in P$,

$$AE_{in}(l) = \begin{cases} \emptyset & \text{if } l = \mathit{init}(P) \\ \bigcap_{(l', l) \in \mathit{flow}(P)} AE_{out}(l') & \end{cases}$$

$$AE_{out}(l) = AE_{in}(l) \setminus kill([i]^l) \cup gen([i]^l)$$

Exercise : available expressions

Build and solve the equation system for the available expression analysis of the following program :

```
[x := a + b]1; [y := a * b]2; while [y > a + b]3 do ([a := a + 1]4; [x := a + b]5)
```

Live variables : a backwards analysis

A variable is *live* if it is used before being redefined.

Domain of values : $LV_{in}(l), LV_{out}(l) \in \mathcal{P}(Var)$

$$kill([x := a]^l) = \{x\}$$

$$kill([\text{skip}]^l) = \emptyset$$

$$kill([b]^l) = \emptyset$$

$$gen([x := a]^l) = \{y \mid y \in Var(a)\}$$

$$gen([\text{skip}]^l) = \emptyset$$

$$gen([b]^l) = \{y \mid y \in Var(b)\}$$

For all $[i]^l \in P$,

$$LV_{in}(l) = LV_{out}(l) \setminus kill([i]^l) \cup gen([i]^l)$$

$$LV_{out}(l) = \begin{cases} \emptyset & \text{if } l \in final(P) \\ \bigcup_{(l,l') \in flow(P)} LV_{in}(l') & \end{cases}$$

Exercise : live variables

Build and solve the equation system for the live variables analysis of the following program :

$[x := 2]^1; [y := 4]^2; [x := 1]^3; (\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6); [x := z]^7$

Analysis classification

The analyses we have presented deal with program execution path.

2 classification criteria :

- ① The information is propagated **forward** or **backward** on paths ;
- ② The property deal with
 - ▶ at least one execution (something **may** happen, use of \cup , least fixpoint)
 - ▶ or all execution (something **must** happen, use of \cap , greatest fixpoint) ;

	Forward analysis	Backward analysis
may	Reachable definitions	Live variables
must	Available expressions	Very busy expressions

Remark : In Abstraction Interpretation we always focus on least fixpoint for a well chosen partial order ($\subseteq, \supseteq, \dots$).

Cooking a dataflow analysis

- 1 Formalize the property you want to track.
- 2 Describe the equation system attached to each program.
 - ▶ forward / backward?
 - ▶ may / must information?
 - ▶ (\cup , least fixpoint) or (\cap , greatest fixpoint)
- 3 Explain why the least/greatest fixpoint exist.
- 4 Explain why Kleene fixpoint iteration will terminate.

References

A modern book, complete, but quite formal : (NNK)

Principle of Program Analysis, F. Nielson, H. Nielson & C. Hankin, Springer.

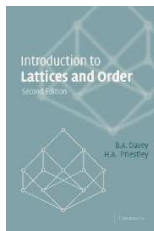
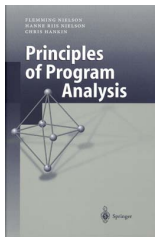
(Elementary) Lattice theory :

Introduction to Lattices and Order, B.A. Davey, & H.A. Priestley, Cambridge Univ

Application to code optimisation : a great classical

Compilers : Principles, Techniques, and Tools, Aho, Sethi & Ullman, Dunod.

Compilateurs : principes, techniques et outils, Aho, Sethi & Ullman, Dunod.



Reading

- ▶ Introduction of NNK (1.1, 1.2, 1.3),
- ▶ Appendix A of NNK,
- ▶ Chapter 2 on dataflow analysis (only section 2.1),
- ▶ ... and complete all left exercises.