Available Expressions Analysis – (forward analysis – largest solution)
$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{AExp}_*), \supseteq, \cap, \mathbf{AExp}_*, \emptyset)$
Reaching Definitions Analysis – (forward analysis – smallest solution)
$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{Var}_* \times \mathbf{Lab}_*^?), \subseteq, \cup, \emptyset, \mathbf{Var}_* \times \mathbf{Lab}_*^?)$
Very Busy Expressions Analysis – (backward analysis – largest solution)
$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{AExp}_*), \supseteq, \cap, \mathbf{AExp}_*, \emptyset)$
Live Variables Analysis – (backward analysis – smallest solution)
$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{Var}_*), \subseteq, \cup, \emptyset, \mathbf{Var}_*)$
Bottom e Top su reticoli completi:
$(P(S), \subseteq) \Rightarrow \bot = \emptyset, \top = S$
$(P(S), \supseteq) \Rightarrow \bot = S, \top = \emptyset$
Interpretazione Astratta
$p \vdash v_1 \rightsquigarrow v_2$ un programma p specifica come un valore v_1 si trasforma in un altro valore v_2.
$p \vdash l_1 \rhd l_2$ un programma p specifica come una proprietà l_1 si trasforma in un altra proprietà l_2.
Relazione di Correttezza
$R : V \times L \rightarrow \{true, false\}$ (è vera quando il valore ha la proprietà indicata)
$(v_1 \; R \; l_1) \wedge (p \vdash v_1 \rightsquigarrow v_2) \wedge (p \vdash l_1 \rhd l_2) \Rightarrow (v_2 \; R \; l_2)$
Nota: (v R l) è come scrivere R(v, l)
proprietà:
$(v \; R \; l_1) \wedge (l_1 \sqsubseteq l_2) \Rightarrow (v \; R \; l_2)$
$(\forall l \in L' \subseteq L : v \; R \; l) \Rightarrow v \; R \; (\sqcap L')$
$(v \; R \; \top)$
$(v \; R \; l_1) \wedge (v \; R \; l_2) \Rightarrow (v \; R \; (l_1 \sqcap l_2))$
Funzione di Rappresentazione
$\beta : V \rightarrow L$ (restituisce la proprietà che meglio descrive il valore)
$(\beta(v_1) \sqsubseteq l_1) \wedge (p \vdash v_1 \rightsquigarrow v_2) \wedge (p \vdash l_1 \rhd l_2) \Rightarrow (\beta(v_2) \sqsubseteq l_2)$
$v \; R_\beta \; l$ iff $\beta(v) \sqsubseteq l$
$\beta_R(v) = \sqcap\{l \mid v \; R \; l\}$
Punti Fissi
$Fix(f) = \{l \mid f(l) = l\}$ (insieme dei punti fissi l)
$Red(f) = \{l \mid f(l) \sqsubseteq l\}$ (f è riduttiva rispetto ad l)
$Ext(f) = \{l \mid f(l) \sqsupseteq l\}$ (f è estensiva rispetto ad l)
$lfp(f) = \sqcap Fix(f) = \sqcap Red(f) \in Fix(f) \subseteq Red(f)$ (least fixed point)
$gfp(f) = \sqcup Fix(f) = \sqcup Ext(f) \in Fix(f) \subseteq Ext(f)$ (greatest fixed point)
$\bot \sqsubseteq f^n(\bot) \sqsubseteq \sqcup_n f^n(\bot) \sqsubseteq lfp(f) \sqsubseteq gfp(f) \sqsubseteq \sqcap_n f^n(\top) \sqsubseteq f^n(\top) \sqsubseteq \top$
Sequenza di elementi $(l^\phi_n)_n$
$\phi : L \times L \rightarrow L$ (funzione totale su l)
$l^\phi_n = l_n$ se $n = 0$
$l^\phi_n = l^\phi_{n-1} \; \phi \; l_n$ se $n > 0$
Operatore di Upper Bound Ŭ
$l_1 \sqsubseteq (l_1 \; \check{U} \; l_2)$
$l_2 \sqsubseteq (l_1 \; \check{U} \; l_2)$
$(l^{\check{U}}_n)_n$ è una catena ascendente $(l^{\check{U}}_n \sqsubseteq l^{\check{U}}_{n+1})$
$l^{\check{U}}_n \sqsupseteq \sqcup\{l_1, l_2, ..., l_n\}$
Operatore di Widening ∇: $L \times L \rightarrow L$
∇ è un operatore di upperbound e la catena ascendente $(l^\nabla_n)_n$ stabilizza
$lfp_\nabla(f) = f^m_\nabla$ (m è il punto nel quale la catena stabilizza: $f^m_\nabla = f^{m+1}_\nabla = ...$)
$l^\nabla_n = l_n$ se $n = 0$
$l^\nabla_n = l^\nabla_{n-1} \nabla l_n$ se $n > 0$
$l^\nabla_n \sqsubseteq l^\nabla_{n+1}$ (catena ascendente)
Operatore di Narrowing Δ: $L \times L \rightarrow L$
$l_2 \sqsubseteq l_1 \Rightarrow l_2 \sqsubseteq (l_1 \Delta l_2) \sqsubseteq l_1$ e la catena discendente $(l^\Delta_n)_n$ stabilizza
$l^\Delta_n = l_n$ se $n = 0$
$l^\Delta_n = l^\Delta_{n-1} \Delta l_n$ se $n > 0$
$l^\Delta_n \sqsupseteq l^\Delta_{n+1}$ (catena discendente)
Connessione di Galois (L, α, γ, M)
connessione tra i due reticoli completi $(L, \sqsubseteq)$ e $(M, \sqsubseteq)$
condizioni:
$\alpha: L \rightarrow M$ e $\gamma: M \rightarrow L$ sono funzioni monotone
$\gamma \circ \alpha \sqsupseteq \lambda l.l$
$\alpha \circ \gamma \sqsubseteq \lambda m.m$
proprietà:
$\gamma(m) = \sqcup\{l \mid \alpha(l) \sqsubseteq m\}$; $\gamma(\top) = \top$; γ è completamente moltiplicativa
$\alpha(l) = \sqcap\{m \mid l \sqsubseteq \gamma(m)\}$; $\alpha(\bot) = \bot$; α è completamente addittiva
$\gamma \circ \alpha \circ \gamma = \gamma$
$\alpha \circ \gamma \circ \alpha = \alpha$

Inserzione di Galois (L, α, γ, M)
connessione tra i due reticoli completi $(L, \sqsubseteq)$ e $(M, \sqsubseteq)$
condizioni:
$\alpha: L \rightarrow M$ e $\gamma: M \rightarrow L$ sono funzioni monotone
$\gamma \circ \alpha \sqsupseteq \lambda l.l$
$\alpha \circ \gamma = \lambda m.m$
proprietà:
α è suriettiva: $\forall m \in M : \exists l \in L : \alpha(l) = m$
γ è iniettiva: $\forall m_1, m_2 \in M : \gamma(m_1) = \gamma(m_2) \Rightarrow m_1 = m_2$
γ rispetta l'ordinamento: $\forall m_1, m_2 \in M : \gamma(m_1) \sqsubseteq \gamma(m_2) \Leftrightarrow m_1 \sqsubseteq m_2$
Adgiunzione
una adjunzione è una connessione di Galois
$\alpha: L \rightarrow M$ e $\gamma: M \rightarrow L$ sono funzioni totali
$\alpha(l) \sqsubseteq m \Leftrightarrow l \sqsubseteq \gamma(m)$
Operatore di riduzione ς: $M \rightarrow M$
rimuove elementi di M in modo tale da trasformare la connessione di Galois (L, α, γ, M) nell'inserzione $(L, \alpha, \gamma, \varsigma(M))$
$\varsigma(m) = \sqcap\{m' \mid \gamma(m) = \gamma(m')\}$
$\varsigma(M) = (\{\varsigma(m) \mid m \in M\}, \sqsubseteq_M)$
$\varsigma(m) = \alpha(\gamma(m))$
$\alpha[L] = \varsigma(M)$
Funzione di estrazione η: $V \rightarrow D$
se $L = (P(D), \subseteq) \Rightarrow (P(V), \alpha_\eta, \gamma_\eta, P(D))$
$\alpha_\eta(V') = \cup\{\beta_\eta(v) \mid v \in V'\} = \{\eta(v) \mid v \in V'\}$
$\gamma_\eta(D') = \{v \in V' \mid \beta_\eta(v) \subseteq D'\} = \{v \mid \eta(v) \in D'\}$
$V' \subseteq V$; $D' \subseteq D$
$\varsigma_\eta(D') = D' \cap \eta[V]$
Relazioni di correttezza
se esiste una connessione di Galois (L, α, γ, M) possiamo scrivere:
$R : V \times L \rightarrow \{true, false\}$ (relazione di correttezza)
$S : V \times M \rightarrow \{true, false\}$
$\beta : V \rightarrow L$ (funzione di rappresentazione)
$v \; S \; m \Leftrightarrow v \; R \; (\gamma(m)) \Leftrightarrow \beta(v) \sqsubseteq \gamma(m) \Leftrightarrow (\alpha \circ \beta)(v) \sqsubseteq m$
Metodo degli attributi indipendenti
$(L_1 \times L_2, \alpha, \gamma, M_1 \times M_2)$
$\alpha(l_1, l_2) = (\alpha(l_1), \alpha(l_2))$
$\gamma(m_1, m_2) = (\gamma(m_1), \gamma(m_2))$
$\alpha(l_1, l_2) \sqsubseteq (m_1, m_2)$
$\Leftrightarrow (\alpha_1(l_1), \alpha_2(l_2)) \sqsubseteq (m_1, m_2)$
$\Leftrightarrow (\alpha_1(l_1) \sqsubseteq m_1) \wedge (\alpha_2(l_2) \sqsubseteq m_2)$
$\Leftrightarrow (l_1 \sqsubseteq \gamma_1(m_1)) \wedge (l_2 \sqsubseteq \gamma_2(m_2))$
$\Leftrightarrow (l_1, l_2) \sqsubseteq (\gamma_1(m_1) \sqcap \gamma_2(m_2))$
$\Leftrightarrow (l_1, l_2) \sqsubseteq \gamma(m_1, m_2)$
Metodo relazionale
combina le seguenti connessioni di galois $(P(V_1), \alpha_1, \gamma_1, P(D_1))$, $(P(V_2), \alpha_2, \gamma_2, P(D_2))$ in $(P(V_1 \times V_2), \alpha, \gamma, P(D_1 \times D_2))$
$\alpha(VV) = \cup\{\alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \mid (v_1, v_2) \in VV\}$ $VV \subseteq V_1 \times V_2$
$\gamma(DD) = \{(v_1, v_2) \mid \alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \subseteq DD\}$ $DD \subseteq D_1 \times D_2$
Spazio delle funzioni totali
sia (L, α, γ, M) una connessione di Galois ed S un insieme
$(S \rightarrow L, \alpha', \gamma', S \rightarrow M)$
$\alpha'(f) = \alpha \circ f$
$\gamma'(g) = \gamma \circ g$
$\gamma'(\alpha'(f)) = \gamma \circ \alpha \circ f \sqsupseteq f$
$\alpha'(\gamma'(g)) = \alpha \circ \gamma \circ g \sqsubseteq g$
Spazio delle funzioni monotone
date due connessioni di Galois $(L_1, \alpha_1, \gamma_1, M_1)$ e $(L_2, \alpha_2, \gamma_2, M_2)$:
$(L_1 \rightarrow L_2, \alpha, \gamma, M_1 \rightarrow M_2)$
$\alpha(f) = \alpha_2 \circ f \circ \gamma_1$
$\gamma(g) = \gamma_2 \circ g \circ \alpha_1$
$\gamma(\alpha(f)) = (\gamma_2 \circ \alpha_2) \circ f \circ (\gamma_1 \circ \alpha_1) \sqsupseteq f$
$\alpha(\gamma(g)) = (\alpha_2 \circ \gamma_2) \circ g \circ (\alpha_1 \circ \gamma_1) \sqsubseteq g$
Operatore di Widening indotto dalla connessione di Galois
dati: (L, α, γ, M) connessione di Galois
e $\nabla_M : M \times M \rightarrow M$ upperbound operator, allora:
$l_1 \nabla_L l_2 = \gamma(\alpha(l_1) \nabla_M \alpha(l_2))$ definisce l'operatore di upper bound $\nabla_L : L \times L \rightarrow L$ che è anche operatore di widening se M soddisfa la condizione di catena ascendente oppure (L, α, γ, M) è un'inserzione di Galois e ∇_M è un operatore di widening.

Available Expressions Analysis – (forward analysis – largest solution)

For each program point determine which expressions must have already been computed, and not later modified, on all paths to the program point.

$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{AExp}_*), \supseteq, \cap, \mathbf{AExp}_*, \emptyset)$

$kill_{AE}$: $\mathbf{Blocks}_* \to P(\mathbf{AExp}_*)$

$kill_{AE}([x := a]^l) = \{a' \in \mathbf{AExp}_* \mid x \in FV(a')\}$

$kill_{AE}([skip]^l) = \emptyset$

$kill_{AE}([b]^l) = \emptyset$

gen_{AE}: $\mathbf{Blocks}_* \to P(\mathbf{AExp}_*)$

$gen_{AE}([x := a]^l) = \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\}$

$gen_{AE}([skip]^l) = \emptyset$

$gen_{AE}([b]^l) = \mathbf{AExp}(b)$

AE_{entry}: $\mathbf{Lab}_* \to P(\mathbf{AExp}_*)$

$AE_{entry}(l) = \emptyset$ if $l = init(S_*)$

$AE_{entry}(l) = \cap\{AE_{exit}(l') \mid (l', l) \in flow(S_*)\}$ otherwise

AE_{exit}: $\mathbf{Lab}_* \to P(\mathbf{AExp}_*)$

$AE_{exit}(l) = (AE_{entry}(l) \setminus kill_{AE}(B^l)) \cup gen_{AE}(B^l)$ where $B^l \in blocks(S_*)$

Very Busy Expressions Analysis – (backward analysis – largest solution)

For each program point determine which expressions must be very busy at the exit from the point.

$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{AExp}_*), \supseteq, \cap, \mathbf{AExp}_*, \emptyset)$

$kill_{VB}$: $\mathbf{Blocks}_* \to P(\mathbf{AExp}_*)$

$kill_{VB}([x := a]^l) = \{a' \in \mathbf{AExp}_* \mid x \in FV(a')\}$

$kill_{VB}([skip]^l) = \emptyset$

$kill_{VB}([b]^l) = \emptyset$

gen_{VB}: $\mathbf{Blocks}_* \to P(\mathbf{AExp}_*)$

$gen_{VB}([x := a]^l) = \mathbf{AExp}(a)$

$gen_{VB}([skip]^l) = \emptyset$

$gen_{VB}([b]^l) = \mathbf{AExp}(b)$

VB_{exit}: $\mathbf{Lab}_* \to P(\mathbf{AExp}_*)$

$VB_{exit}(l) = \emptyset$ if $l = final(S_*)$

$VB_{exit}(l) = \cap\{VB_{entry}(l') \mid (l', l) \in flow^R(S_*)\}$ otherwise

VB_{entry}: $\mathbf{Lab}_* \to P(\mathbf{AExp}_*)$

$VB_{entry}(l) = (VB_{exit}(l) \setminus kill_{VB}(B^l)) \cup gen_{VB}(B^l)$ where $B^l \in blocks(S_*)$

Reaching Definitions Analysis – (forward analysis – smallest solution)

For each program point determine which assignments may have been made and not overwritten, when program execution reaches this point along some path.

$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{Var}_* \times \mathbf{Lab}_*^?), \subseteq, \cup, \emptyset, \mathbf{Var}_* \times \mathbf{Lab}_*^?)$

$kill_{RD}$: $\mathbf{Blocks}_* \to P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$

$kill_{RD}([x := a]^l) = \{(x, ?)\} \cup \{(x, l') \mid B^{l'}$ is an assignment to x in $S_*\}$

$kill_{RD}([skip]^l) = \emptyset$

$kill_{RD}([b]^l) = \emptyset$

gen_{RD}: $\mathbf{Blocks}_* \to P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$

$gen_{RD}([x := a]^l) = \{(x, l)\}$

$gen_{RD}([skip]^l) = \emptyset$

$gen_{RD}([b]^l) = \emptyset$

RD_{entry}: $\mathbf{Lab}_* \to P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$

$RD_{entry}(l) = \{(x, ?) \mid x \in FV(S_*)\}$ if $l = init(S_*)$

$RD_{entry}(l) = \cup\{RD_{exit}(l') \mid (l', l) \in flow(S_*)\}$ otherwise

RD_{exit}: $\mathbf{Lab}_* \to P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$

$RD_{exit}(l) = (RD_{entry}(l) \setminus kill_{RD}(B^l)) \cup gen_{RD}(B^l)$ where $B^l \in blocks(S_*)$

Live Variables Analysis – (backward analysis – smallest solution)

For each program point determine which variables may be live at the exit from the point.

$(L, \sqsubseteq, \sqcup, \bot, \top) = (P(\mathbf{Var}_*), \subseteq, \cup, \emptyset, \mathbf{Var}_*)$

$kill_{LV}$: $\mathbf{Blocks}_* \to P(\mathbf{Var}_*)$

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

$kill_{LV}([skip]^l) = \emptyset$

$kill_{LV}([b]^l) = \emptyset$

gen_{LV}: $\mathbf{Blocks}_* \to P(\mathbf{Var}_*)$

$gen_{LV}([x := a]^l) = FV(a)$

$gen_{LV}([skip]^l) = \emptyset$

$gen_{LV}([b]^l) = FV(b)$

LV_{exit}: $\mathbf{Lab}_* \to P(\mathbf{Var}_*)$

$RD_{exit}(l) = \emptyset$ if $l = final(S_*)$

$RD_{exit}(l) = \cup\{LV_{entry}(l') \mid (l', l) \in flow^R(S_*)\}$ otherwise

LV_{entry}: $\mathbf{Lab}_* \to P(\mathbf{Var}_*)$

$RD_{entry}(l) = (LV_{exit}(l) \setminus kill_{LV}(B^l)) \cup gen_{LV}(B^l)$ where $B^l \in blocks(S_*)$

WHILE Language

Syntactic categories

$a \in$ **AExp**	arithmetic expressions
$b \in$ **BExp**	boolean expressions
$S \in$ **Stmt**	statements
$x, y \in$ **Var**	variables
$n \in$ **Num**	numerals
$l \in$ **Lab**	labels
$op_a \in \mathbf{Op_a}$	arithmetic operators
$op_b \in \mathbf{Op_b}$	boolean operators
$op_r \in \mathbf{Op_r}$	relational operators

Syntax of the language

a ::= x | n | $a_1\ op_a\ a_2$

b ::= true | false | not b | $b_1\ op_b\ b_2$ | $a_1\ op_r\ a_2$

S ::= $[x := a]^1$ | $[\text{skip}]^1$ | S_1 ; S_2 | if $[b]^1$ then S_1 else S_2 | while $[b]^1$ do S

FUN Language

Syntactic categories

e ∈ **Exp**	expressions (or labelled term)
t ∈ **Term**	terms (or unlabelled expressions)
f, x ∈ **Var**	variables
c ∈ **Const**	constants
l ∈ **Lab**	labels
op ∈ **Op**	binary operators

Syntax of the language

$e ::= t^l$

$t ::= c \mid x \mid \text{fn } x \Rightarrow e_0 \mid \text{fun f } x \Rightarrow e_0 \mid e_1\ e_2 \mid \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \mid \text{let } x = e_1 \text{ in e2} \mid e_1\ op\ e_1$

NOTE: *fun f x => e_0* is a recursive variant of *fn x => e_0*

FV : (**Term** ∪ **Exp**) → P(**Var**) free variables

ˆυ ∈ **ˆVal**	= P(**Term**)	abstract values
ˆρ ∈ **ˆEnv**	= **Var** → **ˆVal**	abstract environments: *associate abstract values with each variable*
ˆC ∈ **ˆCache**	= **Lab** → **ˆVal**	abstract caches: *associate abstract values with each labelled program point*

definition points: *points where the funztion abstractions are created*

use points: *points where functions are applied*

Example

```
let f = fn x => x 1
     g = fn y => y+2
     h = fn z => z+3
in (f g) + (f h)
```

- f è la funzione;
- x è il parametro formale di f (i parametri formali sono quelli definiti nella firma di un metodo e vengono trattati, all'interno del metodo, come delle variabili, la cui visibilità termina alla fine del metodo stesso;
- (f g) chiama la funzione f col parametro attuale g che a sua volta è una funzione che ha y come parametro formale; quindi "x 1" è la chiamata alla funzione g con valore 1; il risultato di (f g) è quindi 3.
- (f h) chiama la funzione f col parametro attuale h che a sua volta è una funzione che ha z come parametro formale; quindi "x 1" è la chiamata alla funzione h con valore 1; il risultato di (f h) è quindi 4.
- (f g) + (f h) darà quindi come risultato 3 + 4 = 7.

Functions

init: **Stmt** → **Lab**		initial label of a statement
$init([x := a]^l)$	=	l
$init([\text{skip}]^l)$	=	l
$init(S_1 ; S_2)$	=	$init(S_1)$
$init(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2)$	=	l
$init(\text{while}[b]^l \text{ do } S)$	=	l

final: **Stmt** → P(**Lab**)		final labels in a statement
$final([x := a]^l)$	=	$\{l\}$
$final([\text{skip}]^l)$	=	$\{l\}$
$final(S_1 ; S_2)$	=	$final(S_2)$
$final(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2)$	=	$final(S_1) \cup final(S_2)$
$final(\text{while}[b]^l \text{ do } S)$	=	$\{l\}$

blocks: **Stmt** → P(**Blocks**)		statements or tests associated with a label in a program
$blocks([x := a]^l)$	=	$\{[x := a]^l\}$
$blocks([\text{skip}]^l)$	=	$\{[\text{skip}]^l\}$
$blocks(S_1 ; S_2)$	=	$blocks(S_1) \cup blocks(S_2)$
$blocks(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2)$	=	$\{[b]^l\} \cup blocks(S_1) \cup blocks(S_2)$
$blocks(\text{while}[b]^l \text{ do } S)$	=	$\{[b]^l\} \cup blocks(S)$

labels: **Stmt** → P(**Lab**)		set of labels occurin in a program
$labels(S)$	=	$\{l \mid [B]^l \in blocks(S)\}$

flow: **Stmt** → P(**Lab** x **Lab**)		maps statements to sets of flows
$flow([x := a]^l)$	=	$\emptyset$
$flow([\text{skip}]^l)$	=	$\emptyset$
$flow(S_1 ; S_2)$	=	$flow(S_1) \cup flow(S_2) \cup \{(l, init(S_2)) \mid l \in final(S_1)\}$
$flow(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2)$	=	$flow(S_1) \cup flow(S_2) \cup \{(l, init(S_1)),(l, init(S_2))\}$
$flow(\text{while}[b]^l \text{ do } S)$	=	$flow(S) \cup \{(l, init(S))\} \cup \{(l', l) \mid l' \in final(S)\}$

Instances for the four classical analyses

	Available Expressions	**Reaching Definitions**	**Very Busy Expressions**	**Live Variables**
	Forward Analysis	Forward Analysis	Backward Analysis	Backward Analysis
	Largest solution	Smallest solution	Largest solution	Smallest solution
L	$P(\mathbf{AExp}_*)$	$P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$	$P(\mathbf{AExp}_*)$	$P(\mathbf{Var}_*)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cap$	$\cup$	$\cap$	$\cup$
$\bot$	$\mathbf{AExp}_*$	$\emptyset$	$\mathbf{AExp}_*$	$\emptyset$
$\top$	$\emptyset$	$\mathbf{Var}_* \times \mathbf{Lab}_*^?$	$\emptyset$	$\mathbf{Var}_*$
ι	$\emptyset$	$\{(x,?) \mid x \in FV(S_*)\}$	$\emptyset$	$\emptyset$
E	$\{init(S_*)\}$	$\{init(S_*)\}$	$final(S_*)$	$final(S_*)$
F	$flow(S_*)$	$flow(S_*)$	$flow^R(S_*)$	$flow^R(S_*)$
$\mathscr{F}$	$\{f : L \to L \mid \exists\, l_k, l_g, : f(l) = (l \setminus l_k) \cup l_g\}$			
f_l	$f_l(l) = (l \setminus kill([B]^l)) \cup gen([B]^l)$ where $[B]^l \in blocks(S_*)$			

Available Expressions Analysis

For each program point determine which expressions must have already been computed, and not later modified, on all paths to the program point.

$kill_{\mathrm{AE}}$: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$kill_{\mathrm{AE}}([x := a]^l)$	=	$\{a' \in \mathbf{AExp}_* \mid x \in \mathrm{FV}(a')\}$	
$kill_{\mathrm{AE}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$kill_{\mathrm{AE}}([b]^l)$	=	$\emptyset$	
gen_{AE}: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$gen_{\mathrm{AE}}([x := a]^l)$	=	$\{a' \in \mathbf{AExp}(a) \mid x \notin \mathrm{FV}(a')\}$	
$gen_{\mathrm{AE}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$gen_{\mathrm{AE}}([b]^l)$	=	$\mathbf{AExp}(b)$	
$\mathrm{AE}_{\mathrm{entry}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$\mathrm{AE}_{\mathrm{entry}}(l)$	=	$\emptyset$	if $l = init(S_*)$
$\mathrm{AE}_{\mathrm{entry}}(l)$	=	$\cap\{\mathrm{AE}_{\mathrm{exit}}(l') \mid (l', l) \in flow(S_*)\}$	otherwise
$\mathrm{AE}_{\mathrm{exit}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$\mathrm{AE}_{\mathrm{exit}}(l)$	=	$(\mathrm{AE}_{\mathrm{entry}}(l) \setminus kill_{\mathrm{AE}}(B^l)) \cup \mathrm{gen}_{\mathrm{AE}}(B^l)$ where $B^l \in blocks(S_*)$	

Reaching Definitions Analysis

For each program point determine which assignments may have been made and not overwritten, when program execution reaches this point along some path.

$kill_{RD}$: $\mathbf{Blocks}_* \rightarrow P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$			
$kill_{RD}([x := a]^l)$	=	$\{(x, ?)\} \cup \{(x, l') \mid B^{l'}$ is an assignment to x in $S_*\}$	
$kill_{RD}([\text{skip}]^l)$	=	$\emptyset$	
$kill_{RD}([b]^l)$	=	$\emptyset$	
gen_{RD}: $\mathbf{Blocks}_* \rightarrow P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$			
$gen_{RD}([x := a]^l)$	=	$\{(x, l)\}$	
$gen_{RD}([\text{skip}]^l)$	=	$\emptyset$	
$gen_{RD}([b]^l)$	=	$\emptyset$	
RD_{entry}: $\mathbf{Lab}_* \rightarrow P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$			
$RD_{entry}(l)$	=	$\{(x, ?) \mid x \in FV(S_*)\}$	if $l = init(S_*)$
$RD_{entry}(l)$	=	$\cup\{RD_{exit}(l') \mid (l', l) \in flow(S_*)\}$	otherwise
RD_{exit}: $\mathbf{Lab}_* \rightarrow P(\mathbf{Var}_* \times \mathbf{Lab}_*^?)$			
$RD_{exit}(l)$	=	$(RD_{entry}(l) \setminus kill_{RD}(B^l)) \cup gen_{RD}(B^l)$ where $B^l \in blocks(S_*)$	

Very Busy Expressions Analysis

For each program point determine which expressions must be very busy at the exit from the point.

$kill_{\mathrm{VB}}$: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$kill_{\mathrm{VB}}([x := a]^l)$	=	$\{a' \in \mathbf{AExp}_* \mid x \in \mathrm{FV}(a')\}$	
$kill_{\mathrm{VB}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$kill_{\mathrm{VB}}([b]^l)$	=	$\emptyset$	
gen_{VB}: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$gen_{\mathrm{VB}}([x := a]^l)$	=	$\mathbf{AExp}(a)$	
$gen_{\mathrm{VB}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$gen_{\mathrm{VB}}([b]^l)$	=	$\mathbf{AExp}(b)$	
$\mathrm{VB}_{\mathrm{exit}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$\mathrm{VB}_{\mathrm{exit}}(l)$	=	$\emptyset$	if $l = final(S_*)$
$\mathrm{VB}_{\mathrm{exit}}(l)$	=	$\cap\{\mathrm{VB}_{\mathrm{entry}}(l') \mid (l', l) \in flow^R(S_*)\}$	otherwise
$\mathrm{VB}_{\mathrm{entry}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{AExp}_*)$			
$\mathrm{VB}_{\mathrm{entry}}(l)$	=	$(\mathrm{VB}_{\mathrm{exit}}(l) \setminus kill_{\mathrm{VB}}(B^l)) \cup \mathrm{gen}_{\mathrm{VB}}(B^l)$ where $B^l \in blocks(S_*)$	

Live Variables Analysis

For each program point determine which variables may be live at the exit from the point.

$kill_{\mathrm{LV}}$: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{Var}_*)$			
$kill_{\mathrm{LV}}([x := a]^l)$	=	$\{x\}$	
$kill_{\mathrm{LV}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$kill_{\mathrm{LV}}([b]^l)$	=	$\emptyset$	
gen_{LV}: $\mathbf{Blocks}_* \to \mathrm{P}(\mathbf{Var}_*)$			
$gen_{\mathrm{LV}}([x := a]^l)$	=	$\mathrm{FV}(a)$	
$gen_{\mathrm{LV}}([\mathrm{skip}]^l)$	=	$\emptyset$	
$gen_{\mathrm{LV}}([b]^l)$	=	$\mathrm{FV}(b)$	
$\mathrm{LV}_{\mathrm{exit}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{Var}_*)$			
$\mathrm{RD}_{\mathrm{exit}}(l)$	=	$\emptyset$	if $l = final(S_*)$
$\mathrm{RD}_{\mathrm{exit}}(l)$	=	$\cup\{\mathrm{LV}_{\mathrm{entry}}(l') \mid (l', l) \in flow^R(S_*)\}$	otherwise
$\mathrm{LV}_{\mathrm{entry}}$: $\mathbf{Lab}_* \to \mathrm{P}(\mathbf{Var}_*)$			
$\mathrm{RD}_{\mathrm{entry}}(l)$	=	$(\mathrm{LV}_{\mathrm{exit}}(l) \setminus kill_{\mathrm{LV}}(B^l)) \cup \mathrm{gen}_{\mathrm{LV}}(B^l)$ where $B^l \in blocks(S_*)$	

Abstract Control Flow Analysis

[con] $(\hat{C}, \hat{\rho}) \vDash c^{l}$

iff $\emptyset \subseteq \hat{C}(l)$ *sempre verificata*

[var] $(\hat{C}, \hat{\rho}) \vDash x^{l}$

iff $\hat{\rho}(x) \subseteq \hat{C}(l)$

[fn] $(\hat{C}, \hat{\rho}) \vDash (\text{fn } x \Rightarrow e_0)^{l}$

iff $\{\text{fn } x \Rightarrow e_0\} \subseteq \hat{C}(l)$

[fun] $(\hat{C}, \hat{\rho}) \vDash (\text{fun } f\ x \Rightarrow e_0)^{l}$

iff $\{\text{fun } f\ x \Rightarrow e_0\} \subseteq \hat{C}(l)$

[app] $(\hat{C}, \hat{\rho}) \vDash (t_1^{l1}\ t_2^{l2})^{l}$

iff $(\hat{C}, \hat{\rho}) \vDash t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \vDash t_2^{l2} \wedge$

$(\ \forall(\text{fn } x \Rightarrow t_0^{l0}) \in \hat{C}(l_1) : (\hat{C}, \hat{\rho}) \vDash t_0^{l0} \wedge \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l)\ ) \wedge$

$(\ \forall(\text{fun } f\ x \Rightarrow t_0^{l0}) \in \hat{C}(l_1) : (\hat{C}, \hat{\rho}) \vDash t_0^{l0} \wedge \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l) \wedge \{\text{fun } f\ x \Rightarrow t_0^{l0}\} \subseteq \hat{\rho}(f)\ )$

[if] $(\hat{C}, \hat{\rho}) \vDash (\text{if } t_0^{l0} \text{ then } t_1^{l1} \text{ else } t_2^{l2})^{l}$

iff $(\hat{C}, \hat{\rho}) \vDash t_0^{l0} \wedge (\hat{C}, \hat{\rho}) \vDash t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \vDash t_2^{l2} \wedge \hat{C}(l_1) \subseteq \hat{C}(l) \wedge \hat{C}(l_2) \subseteq \hat{C}(l)$

[let] $(\hat{C}, \hat{\rho}) \vDash (\text{let } x = t_1^{l1} \text{ in } t_2^{l2})^{l}$

iff $(\hat{C}, \hat{\rho}) \vDash t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \vDash t_2^{l2} \wedge \hat{C}(l_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_2) \subseteq \hat{C}(l)$

[op] $(\hat{C}, \hat{\rho}) \vDash (t_1^{l1}\ \text{op}\ t_2^{l2})^{l}$

iff $(\hat{C}, \hat{\rho}) \vDash t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \vDash t_2^{l2}$

Syntax directed Control Flow Analysis

[con] $(\hat{C}, \hat{\rho}) \models_s c^l$

iff $\emptyset \subseteq \hat{C}(l)$ *sempre verificata*

[var] $(\hat{C}, \hat{\rho}) \models_s x^l$

iff $\hat{\rho}(x) \subseteq \hat{C}(l)$

[fn] $(\hat{C}, \hat{\rho}) \models_s (\text{fn } x \Rightarrow e_0)^l$

iff $\{\text{fn } x \Rightarrow e_0\} \subseteq \hat{C}(l) \wedge (\hat{C}, \hat{\rho}) \models_s e_0$

[fun] $(\hat{C}, \hat{\rho}) \models_s (\text{fun } f\ x \Rightarrow e_0)^l$

iff $\{\text{fun } f\ x \Rightarrow e_0\} \subseteq \hat{C}(l) \wedge (\hat{C}, \hat{\rho}) \models_s e_0 \wedge \{\text{fun } f\ x \Rightarrow e_0\} \subseteq \hat{\rho}(f)$

[app] $(\hat{C}, \hat{\rho}) \models_s (t_1^{l1}\ t_2^{l2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l2} \wedge$

$(\ \forall(\text{fn } x \Rightarrow t_0^{l0}) \in \hat{C}(l_1) : \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l)\ ) \wedge$

$(\ \forall(\text{fun } f\ x \Rightarrow t_0^{l0}) \in \hat{C}(l_1) : \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l)\ )$

[if] $(\hat{C}, \hat{\rho}) \models_s (\text{if } t_0^{l0} \text{ then } t_1^{l1} \text{ else } t_2^{l2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_0^{l0} \wedge (\hat{C}, \hat{\rho}) \models_s t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l2} \wedge \hat{C}(l_1) \subseteq \hat{C}(l) \wedge \hat{C}(l_2) \subseteq \hat{C}(l)$

[let] $(\hat{C}, \hat{\rho}) \models_s (\text{let } x = t_1^{l1} \text{ in } t_2^{l2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l2} \wedge \hat{C}(l_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_2) \subseteq \hat{C}(l)$

[op] $(\hat{C}, \hat{\rho}) \models_s (t_1^{l1}\ \text{op}\ t_2^{l2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l2}$

Contraint based Control Flow Analysis

[con] $C_*\langle c^1\rangle = \emptyset$

[var] $C_*\langle x^1\rangle = \{r(x) \subseteq C(l)\}$

[fn] $C_*\langle(\text{fn } x \Rightarrow e_0)^1\rangle = \{\{\text{fn } x \Rightarrow e_0\} \subseteq C(l)\} \cup C_*\langle e_0\rangle$

[fun] $C_*\langle(\text{fun } f\ x \Rightarrow e_0)^1\rangle = \{\{\text{fun } f\ x \Rightarrow e_0\} \subseteq C(l)\} \cup C_*\langle e_0\rangle \cup \{\{\text{fun } f\ x \Rightarrow e_0\} \subseteq r(f)\}$

[app] $C_*\langle(t_1^{11}\ t_2^{12})^1\rangle = C_*\langle t_1^{11}\rangle \cup C_*\langle t_2^{12}\rangle$

$\cup \{\{t\} \subseteq C(l_1) \Rightarrow C(l_2) \subseteq r(x) \mid t = (\text{fn } x \Rightarrow t_0^{10}) \in \mathbf{Term}_*\}$

$\cup \{\{t\} \subseteq C(l_1) \Rightarrow C(l_0) \subseteq C(l) \mid t = (\text{fn } x \Rightarrow t_0^{10}) \in \mathbf{Term}_*\}$

$\cup \{\{t\} \subseteq C(l_1) \Rightarrow C(l_2) \subseteq r(x) \mid t = (\text{fun } f\ x \Rightarrow t_0^{10}) \in \mathbf{Term}_*\}$

$\cup \{\{t\} \subseteq C(l_1) \Rightarrow C(l_0) \subseteq C(l) \mid t = (\text{fun } f\ x \Rightarrow t_0^{10}) \in \mathbf{Term}_*\}$

[if] $C_*\langle(\text{if } t_0^{10} \text{ then } t_1^{11} \text{ else } t_2^{12})^1\rangle = C_*\langle t_0^{10}\rangle \cup C_*\langle t_1^{11}\rangle \cup C_*\langle t_2^{12}\rangle \cup \{C(l_1) \subseteq C(l)\} \cup \{C(l_2) \subseteq C(l)\}$

[let] $C_*\langle(\text{let } x = t_1^{11} \text{ in } t_2^{12})^1\rangle = C_*\langle t_1^{11}\rangle \cup C_*\langle t_2^{12}\rangle \cup \{C(l_1) \subseteq r(x)\} \cup \{C(l_2) \subseteq r(x)\}$

[op] $C_*\langle(t_1^{11}\ \text{op}\ t_2^{12})^1\rangle = C_*\langle t_1^{11}\rangle \cup C_*\langle t_2^{12}\rangle$