Script generated by TTT

Title: Seidl: Programmoptimierung (18.12.2013)

Date: Wed Dec 18 08:31:25 CET 2013

Duration: 87:43 min

Pages: 71

2.3 Procedures

We extend our mini-programming language by procedures without parameters and procedure calls.

For that, we introduce a new statement:

$$f();$$

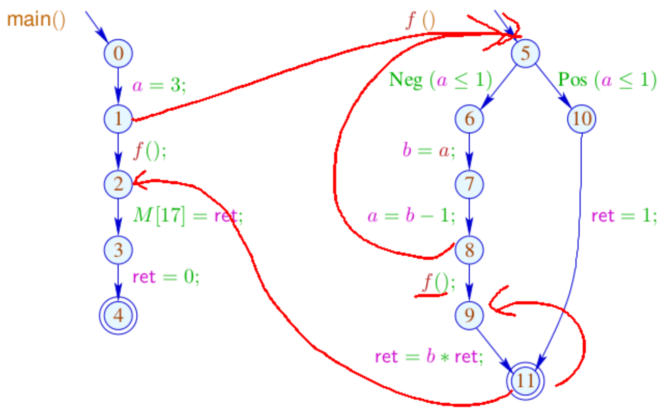
Every procedure f has a definition:

$$f\,()\ \{\ stmt^*\ \}$$

Additionally, we distinguish between global and local variables.

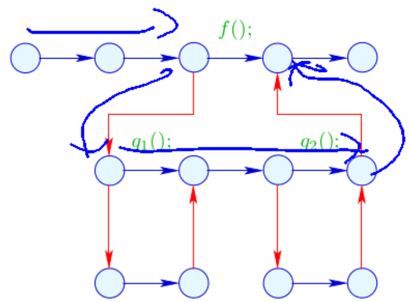
Program execution starts with the call of a procedure main () .

... in the Example:

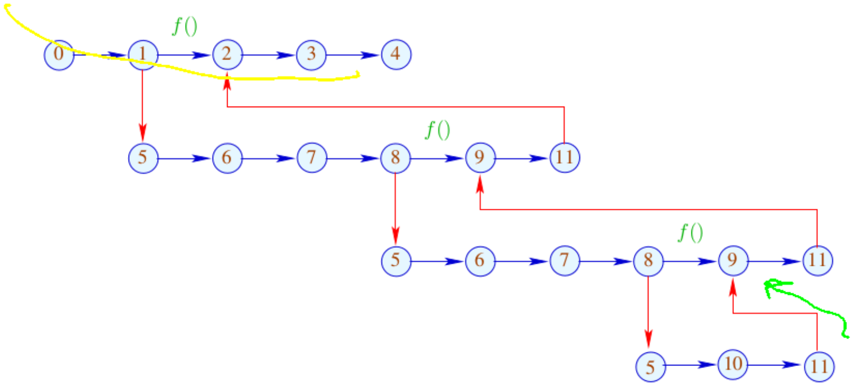


In order to optimize such programs, we require an extended operational semantics ;-)

Program executions are no longer paths, but forests:



... in the Example:



The function $[\![.]\!]$ is extended to computation forests: w :

$$[\![w]\!] : (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z}) \to (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})$$

For a call $k = (u, f();, v)$ we must:

- determine the initial values for the locals:

$$\mathsf{enter}\ \rho = \{x \mapsto 0 \mid x \in \mathit{Locals}\} \oplus (\rho|_{\mathit{Globals}})$$

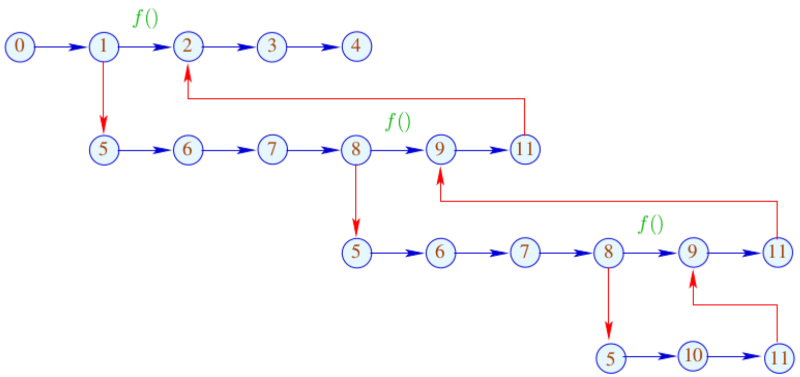
- ... combine the new values for the globals with the old values for the locals:

$$\mathsf{combine}\ (\rho_1, \rho_2) = (\rho_1|_{\mathit{Locals}}) \oplus (\rho_2|_{\mathit{Globals}})$$

- ... evaluate the computation forest inbetween:

$$\begin{array}{lll} [\![k\ \langle w \rangle]\!]\ (\rho, \mu) & = & \mathsf{let}\ \ (\rho_1, \mu_1) = [\![w]\!]\ (\mathsf{enter}\ \rho, \mu) \\ & & \mathsf{in}\ \ \ (\mathsf{combine}\ (\rho, \rho_1), \mu_1) \end{array}$$

... in the Example:



The function $[\![.]\!]$ is extended to computation forests: w :

$$[\![w]\!] : (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z}) \to (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})$$

For a call $k = (u, f();, v)$ we must:

- determine the initial values for the locals:

$$\mathsf{enter}\ \rho = \{x \mapsto 0 \mid x \in \mathit{Locals}\} \oplus (\rho|_{\mathit{Globals}})$$

- ... combine the new values for the globals with the old values for the locals:

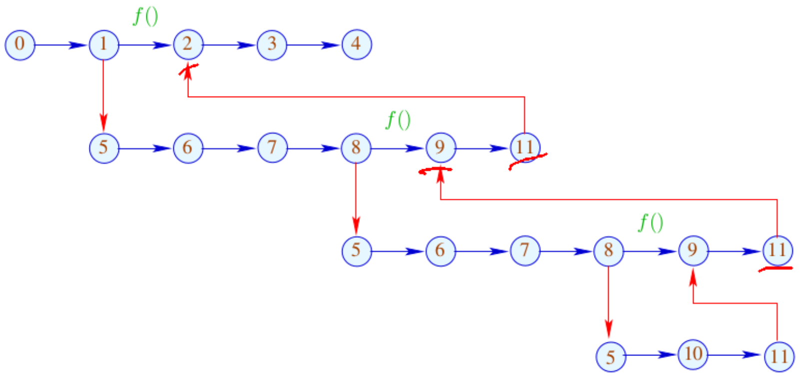
$$\mathsf{combine}\ (\rho_1, \rho_2) = (\rho_1|_{\mathit{Locals}}) \oplus (\rho_2|_{\mathit{Globals}})$$

- ... evaluate the computation forest inbetween:

$$\begin{array}{lll} [\![k\ \langle w \rangle]\!]\ (\rho, \mu) & = & \mathsf{let}\ \ (\rho_1, \mu_1) = [\![w]\!]\ (\mathsf{enter}\ \rho, \mu) \\ & & \mathsf{in}\ \ \ (\mathsf{combine}\ (\rho, \rho_1), \mu_1) \end{array}$$

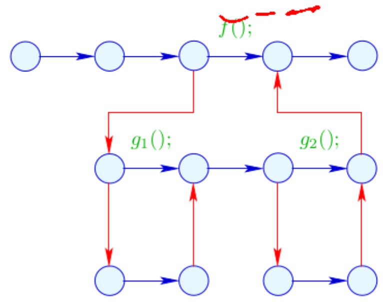
<k> w </k>

... in the Example:



In order to optimize such programs, we require an extended operational semantics ;-)

Program executions are no longer paths, but forests:



The function $[\![.]\!]$ is extended to computation forests: w :

$$[\![w]\!] : (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z}) \to (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})$$

For a call $k = (u, f();, v)$ we must:

- determine the initial values for the locals:

$$\mathsf{enter}\ \rho = \{x \mapsto 0 \mid x \in \mathit{Locals}\} \oplus (\rho|_{\mathit{Globals}})$$

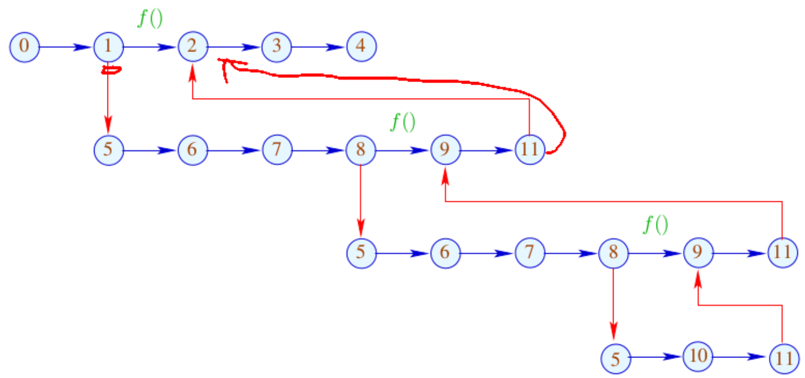
- ... combine the new values for the globals with the old values for the locals:

$$\mathsf{combine}\ (\rho_1, \rho_2) = (\rho_1|_{\mathit{Locals}}) \oplus (\rho_2|_{\mathit{Globals}})$$

- ... evaluate the computation forest inbetween:

$$\begin{array}{lll} [\![k\ \langle w \rangle]\!]\ (\rho, \mu) & = & \mathsf{let}\ \ (\rho_1, \mu_1) = [\![w]\!]\ (\mathsf{enter}\ \rho, \mu) \\ & & \mathsf{in}\ \ \ (\mathsf{combine}\ (\rho, \rho_1), \mu_1) \end{array}$$

... in the Example:



The function $[\![.]\!]$ is extended to computation forests: w :

$$[\![w]\!] : (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z}) \to (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})$$

For a call $k = (u, f();, v)$ we must:

- determine the initial values for the locals:

$$\mathsf{enter}\ \rho = \{x \mapsto 0 \mid x \in \mathit{Locals}\} \oplus (\rho|_{\mathit{Globals}})$$

- ... combine the new values for the globals with the old values for the locals:

$$\mathsf{combine}\ (\rho_1, \rho_2) = (\rho_1|_{\mathit{Locals}}) \oplus (\rho_2|_{\mathit{Globals}})$$

- ... evaluate the computation forest inbetween:

$$\begin{array}{lll} [\![k\ \langle w \rangle]\!]\ (\rho, \mu) & = & \mathsf{let}\ (\rho_1, \mu_1) = [\![w]\!]\ (\mathsf{enter}\ \rho, \mu) \\ & & \mathsf{in}\ \ (\mathsf{combine}\ (\rho, \rho_1), \mu_1) \end{array}$$

The function $[\![.]\!]$ is extended to computation forests: w :

$$[\![w]\!] : (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z}) \to (\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})$$

For a call $k = (u, f();, v)$ we must:

- determine the initial values for the locals:

$$\mathsf{enter}\ \rho = \{x \mapsto 0 \mid x \in \mathit{Locals}\} \oplus (\rho|_{\mathit{Globals}})$$

- ... combine the new values for the globals with the old values for the locals:

$$\mathsf{combine}\ (\rho_1, \rho_2) = (\rho_1|_{\mathit{Locals}}) \oplus (\rho_2|_{\mathit{Globals}})$$

- ... evaluate the computation forest inbetween:

$$\begin{array}{lll} [\![k\ \langle w \rangle]\!]\ (\rho, \mu) & = & \mathsf{let}\ (\rho_1, \mu_1) = [\![w]\!]\ (\mathsf{enter}\ \rho, \mu) \\ & & \mathsf{in}\ \ (\mathsf{combine}\ (\rho, \rho_1), \mu_1) \end{array}$$

525

Configurations:

$$\begin{array}{lcl} \mathit{configuration} & == & \mathit{stack} \times \mathit{store} \\ \mathit{store} & == & \mathit{globals} \times (\mathbb{N} \to \mathbb{Z}) \\ \mathit{globals} & == & (\mathit{Globals} \to \mathbb{Z}) \\ \mathit{stack} & == & \mathit{frame} \cdot \mathit{frame}^* \\ \mathit{frame} & == & \mathit{point} \times \mathit{locals} \\ \mathit{locals} & == & (\mathit{Locals} \to \mathbb{Z}) \end{array}$$

A *frame* specifies the local state of computation inside a procedure call :-)

The leftmost frame corresponds to the current call.

Configurations:

$$\begin{array}{lcl} \mathit{configuration} & == & \mathit{stack} \times \mathit{store} \\ \mathit{store} & == & \mathit{globals} \times (\mathbb{N} \to \mathbb{Z}) \\ \mathit{globals} & == & (\mathit{Globals} \to \mathbb{Z}) \\ \mathit{stack} & == & \mathit{frame} \cdot \mathit{frame}^* \\ \mathit{frame} & == & \mathit{point} \times \mathit{locals} \\ \mathit{locals} & == & (\mathit{Locals} \to \mathbb{Z}) \end{array}$$

A *frame* specifies the local state of computation inside a procedure call :-)

The leftmost frame corresponds to the current call.

527

Computation steps refer to the current call :-)

The novel kinds of steps:

call $k = (u, f\,();, v)$:

$$(\boxed{(u,\rho)} \cdot \sigma, \langle\gamma,\mu\rangle) \Longrightarrow (\boxed{(u_f, \{x \to 0 \mid x \in Locals\}) \cdot (v,\rho)} \cdot \sigma, \langle\gamma,\mu\rangle)$$

u_f entry point of f

return:

$$(\boxed{(r_f, \_)} \cdot \sigma, \langle\gamma,\mu\rangle) \Longrightarrow (\sigma, \langle\gamma,\mu\rangle)$$

r_f return point of f

Computation steps refer to the current call :-)

The novel kinds of steps:

call $k = (u, f\,();, v)$:

$$(\boxed{(u,\rho)} \cdot \sigma, \langle\gamma,\mu\rangle) \Longrightarrow (\boxed{(u_f, \{x \to 0 \mid x \in Locals\}) \cdot (v,\rho)} \cdot \sigma, \langle\gamma,\mu\rangle)$$

u_f entry point of f

return:

$$(\boxed{(r_f, \_)} \cdot \sigma, \langle\gamma,\mu\rangle) \Longrightarrow (\sigma, \langle\gamma,\mu\rangle)$$

r_f return point of f

The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:

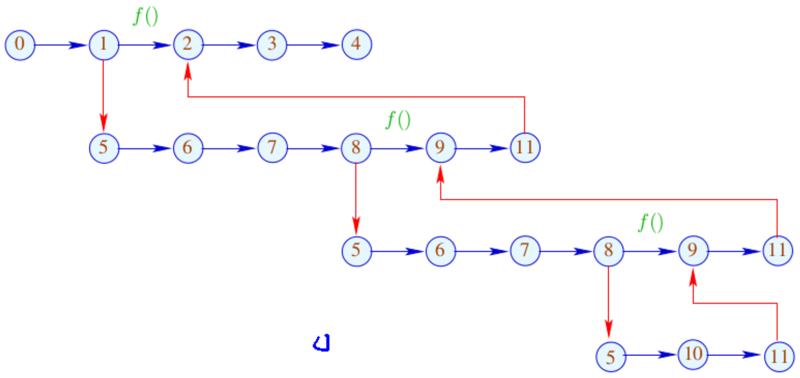
1	

The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:

5	$b \mapsto 0$
2	

... in the Example:

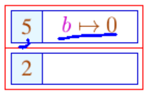


The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:

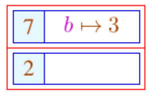
The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:



The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:



The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:

5	$b \mapsto 0$
9	$b \mapsto 3$
2	

The call stack explicitly implements the DFS traversal through the computation forest :-)

... in the Example:

7	$b \mapsto 2$
9	$b \mapsto 3$
2	

1. Idea: Inlining

Copy the procedure body at every call site !!!

Example:

$abs\ ()\ \{$
$\quad a_2 = -a_1;$
$\quad max\ ();$
$\}$

$max\ ()\ \{$
$\quad \text{if}\ \ (a_1 < a_2)\ \ \{\ \text{ret} = a_2;\ \text{goto}\ \_exit;\ \}$
$\quad \text{ret} = a_1;$
$\_exit\ :$
$\}$

... yields:

$abs\ ()\ \{$
$a_2 = -a_1;$
$\quad \text{if}\ \ (a_1 < a_2)\ \ \{\ \text{ret} = a_2;\ \text{goto}\ \_exit;\ \}$
$\quad \text{ret} = a_1;$
$\_exit\ :$
$\}$

Problems:

- The copied block may modify the locals of the calling procedure ???
- More general: Multiple use of local variable names may lead to errors.
- Multiple calls of a procedure may lead to code duplication :-((
- How can we handle recursion ???

1. Idea: Inlining

Copy the procedure body at every call site !!!

Example:

abs () {
  $a_2 = -a_1$;
  max ();
}

max () {
  if $(a_1 < a_2)$ { ret $= a_2$; goto $\_exit$; }
  ret $= a_1$;
$\_exit$:
}

... yields:

abs () {
$a_2 = -a_1$;
  if $(a_1 < a_2)$ { ret $= a_2$; goto $\_exit$; }
  ret $= a_1$;
$\_exit$:
}

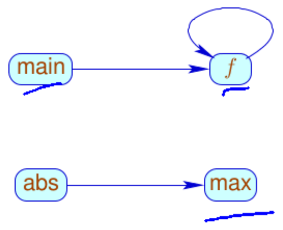
Problems:

- The copied block may modify the locals of the calling procedure ???
- More general: Multiple use of local variable names may lead to errors.
- Multiple calls of a procedure may lead to code duplication :-((
- How can we handle recursion ???

Detection of Recursion:

We construct the call-graph of the program.

In the Examples:



Call-Graph:

- The nodes are the procedures.
- An edge connexts g with h, whenever the body of g contains a call of h.

Strategies for Inlining:

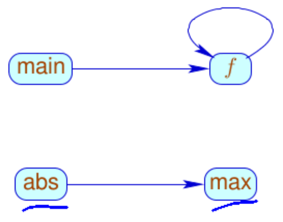
- Just copy nur leaf-procedures, i.e., procedures without further calls :-)
- Copy all non-recursive procedures!

... here, we consider just leaf-procedures ;-)

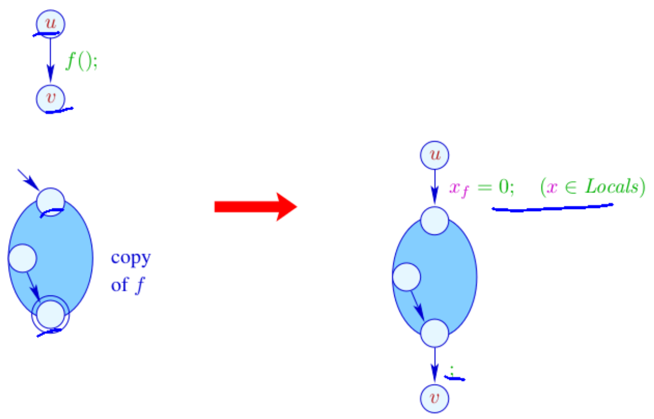
Detection of Recursion:

We construct the call-graph of the program.

In the Examples:



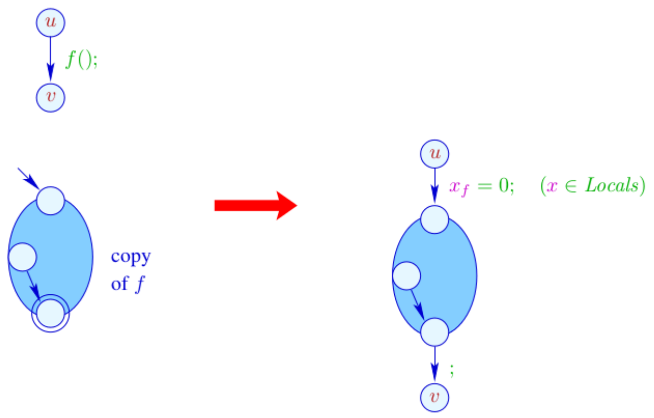
Transformation 9:



Note:

- The Nop-edge can be eliminated if the *stop*-node of f has no out-going edges ...
- The x_f are the copies of the locals of the procedure f.
- According to our semantics of procedure calls, these must be initialized with 0 :-)

Transformation 9:



Note:

- The Nop-edge can be eliminated if the *stop*-node of f has no out-going edges ...
- The x_f are the copies of the locals of the procedure f.
- According to our semantics of procedure calls, these must be initialized with 0 :-)

548

2. Idea: Elimination of Tail Recursion

$$
\begin{array}{l}
f\,()\ \{\quad \mathsf{int}\ b;\\
\qquad \mathsf{if}\ (a_2 \leq 1)\ \{\ \mathsf{ret} = a_1;\ \mathsf{goto}\ \_exit;\ \}\\
\qquad b = a_1 \cdot a_2;\\
\qquad a_2 = a_2 - 1;\\
\qquad a_1 = b;\\
\qquad f\,();\\
\_exit\ :\\
\}
\end{array}
$$

ret = ret

After the procedure call, nothing in the body remains to be done.

$\Longrightarrow$ We may directly jump to the beginning :-)

... after having reset the locals to 0.

... this yields in the Example:

$f\,()\;\{\quad \mathsf{int}\;\; b;$

$\_f:\quad \mathsf{if}\;\; (a_2 \leq 1)\;\; \{\;\; \mathsf{ret} = a_1;\; \mathsf{goto}\;\; \_exit;\;\; \}$

$\quad b = a_1 \cdot a_2;$

$\quad a_2 = a_2 - 1;$

$\quad a_1 = b;$

$\quad \cancel{b = 0;}\;\; \mathsf{goto}\;\; \_f;$

$\_exit:$

$\}$

// It works, since we have ruled out references to variables!

Transformation 11:

Warning:

- → This optimization is crucial for programming languages without iteration constructs !!!
- → Duplication of code is not necessary :-)
- → No variable renaming is necessary :-)
- → The optimization may also be profitable for non-recursive tail calls :-)
- → The corresponding code may contain jumps from the body of one procedure into the body of another ???

Warning:

- → This optimization is crucial for programming languages without iteration constructs !!!
- → Duplication of code is not necessary :-)
- → No variable renaming is necessary :-)
- → The optimization may also be profitable for non-recursive tail calls :-)
- → The corresponding code may contain jumps from the body of one procedure into the body of another ???

2. Idea: Elimination of Tail Recursion

$$
\begin{array}{l}
f\,()\ \{\quad \texttt{int}\ b;\\
\qquad \texttt{if}\ (a_2 \leq 1)\ \{\ \texttt{ret} = a_1;\ \texttt{goto}\ \_exit;\ \ \}\\
\qquad b = a_1 \cdot a_2;\\
\qquad a_2 = a_2 - 1;\\
\qquad a_1 = b;\\
\qquad f\,();\\
\_exit\ :\\
\}
\end{array}
$$

After the procedure call, nothing in the body remains to be done.

$\Longrightarrow$ We may directly jump to the beginning :-)

... after having reset the locals to 0.

... this yields in the Example:

$$
\begin{array}{ll}
f\,()\ \{\quad \texttt{int}\ b; & \\
\_f\ : & \texttt{if}\ (a_2 \leq 1)\ \{\ \texttt{ret} = a_1;\ \texttt{goto}\ \_exit;\ \ \}\\
 & b = a_1 \cdot a_2;\\
 & a_2 = a_2 - 1;\\
 & a_1 = b;\\
 & b = 0;\ \ \texttt{goto}\ \ \_f;\\
\_exit : & \\
\} &
\end{array}
$$

g

// It works, since we have ruled out references to variables!

Warning:

- → This optimization is crucial for programming languages without iteration constructs !!!
- → Duplication of code is not necessary :-)
- → No variable renaming is necessary :-)
- → The optimization may also be profitable for non-recursive tail calls :-)
- → The corresponding code may contain jumps from the body of one procedure into the body of another ???

Background 4: Interprocedural Analysis

So far, we can analyze each procedure separately.

- → The costs are moderate :-)
- → The methods also work in presence of separate compilation :-)
- → At procedure calls, we must assume the worst case :-(
- → Constant propagation only works for local constants :-((

Question:

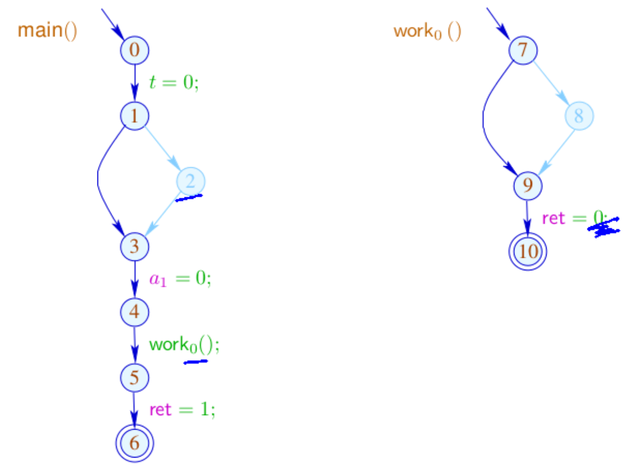
How can recursive programs be analyzed ???

Example: Constant Propagation

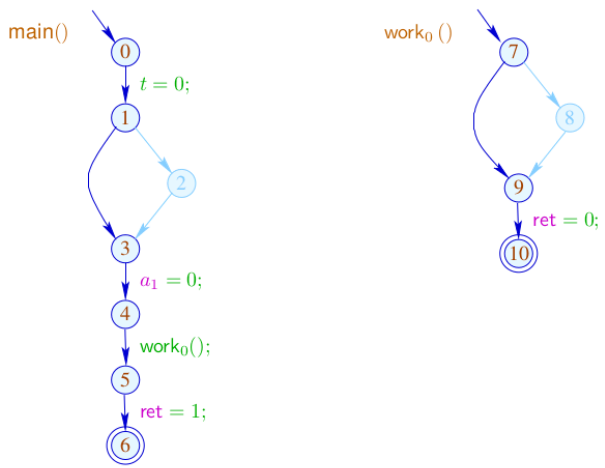
main() { **int** t;
  $t = 0$;
  if (t) ~~$M[17]$~~ $= 3$;
  $a_1 = t$;   0
  work ();
  ret $= 1 -$ ret;   0
}

work() {   0
  if (a_1) ~~work()~~;
  ret $= a_1$;   0
}

Example: Constant Propagation



Example: Constant Propagation



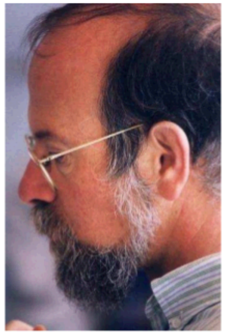
(1) Functional Approach:

Let $\mathbb{D}$ denote a complete lattice of (abstract) states.

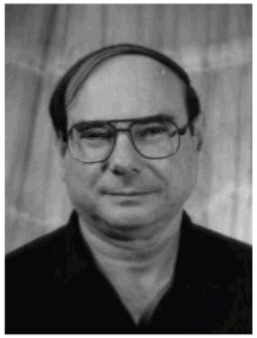
Idea:

Represent the effect of $f()$ by a function:

$$[\![f]\!]^\sharp \; : \; \mathbb{D} \to \mathbb{D}$$



Micha Sharir, Tel Aviv University



Amir Pnueli, Weizmann Institute

In order to determine the effect of a call edge $k = (u, f\,();, v)$ we require abstract functions:

$$\begin{array}{lll} \mathsf{enter}^\sharp & : & \mathbb{D} \to \mathbb{D} \\ \mathsf{combine}^\sharp & : & \mathbb{D}^2 \to \mathbb{D} \end{array}$$

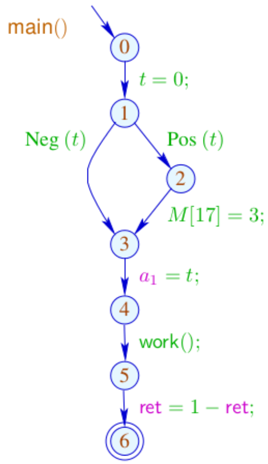
Then we define:

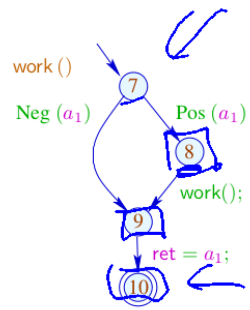
$$[\![k]\!]^\sharp\, D \quad = \quad \mathsf{combine}^\sharp\, (D, [\![f]\!]^\sharp\, (\mathsf{enter}^\sharp\, D))$$

... for Constant Propagation:

$$\begin{array}{lll} \mathbb{D} & = & (\mathit{Vars} \to \mathbb{Z}^\top)_\bot \\ \mathsf{enter}^\sharp\, D & = & \begin{cases} \bot & \text{if} \quad D = \bot \\ D|_{\mathit{Globals}} \oplus \{x \mapsto 0 \mid x \in \mathit{Locals}\} & \text{otherwise} \end{cases} \\ \mathsf{combine}^\sharp\, (D_1, D_2) & = & \begin{cases} \bot & \text{if} \quad D_1 = \bot \vee D_2 = \bot \\ D_1|_{\mathit{Locals}} \oplus D_2|_{\mathit{Globals}} & \text{otherwise} \end{cases} \end{array}$$

Example: Constant Propagation





$[\![\text{work}]\!]^\sharp =$

(1) **Functional Approach**:

Let $\mathbb{D}$ denote a complete lattice of (abstract) states.

Idea:

Represent the effect of $f()$ by a function:

$$[\![f]\!]^\sharp \; : \; \mathbb{D} \to \mathbb{D}$$

... for Constant Propagation:

$$\mathbb{D} = (\mathit{Vars} \to \mathbb{Z}^\top)_\bot$$

$$\mathsf{enter}^\sharp\; D = \begin{cases} \bot & \text{if} \quad D = \bot \\ D|_{\mathit{Globals}} \oplus \{x \mapsto 0 \mid x \in \mathit{Locals}\} & \text{otherwise} \end{cases}$$

$$\mathsf{combine}^\sharp\; (D_1, D_2) = \begin{cases} \bot & \text{if} \quad D_1 = \bot \vee D_2 = \bot \\ D_1|_{\mathit{Locals}} \oplus D_2|_{\mathit{Globals}} & \text{otherwise} \end{cases}$$

In order to determine the effect of a call edge $k = (u, f\,();, v)$ we require abstract functions:

$$\begin{array}{lll} \mathsf{enter}^\sharp & : & \mathbb{D} \to \mathbb{D} \\ \mathsf{combine}^\sharp & : & \mathbb{D}^2 \to \mathbb{D} \end{array}$$

Then we define:

$$[\![k]\!]^\sharp\; D = \mathsf{combine}^\sharp\; (D, [\![f]\!]^\sharp\; (\mathsf{enter}^\sharp\; D))$$

... for Constant Propagation:

$$\mathbb{D} = (\mathit{Vars} \to \mathbb{Z}^\top)_\bot$$

$$\mathsf{enter}^\sharp\; D = \begin{cases} \bot & \text{if} \quad D = \bot \\ D|_{\mathit{Globals}} \oplus \{x \mapsto 0 \mid x \in \mathit{Locals}\} & \text{otherwise} \end{cases}$$

$$\mathsf{combine}^\sharp\; (D_1, D_2) = \begin{cases} \bot & \text{if} \quad D_1 = \bot \vee D_2 = \bot \\ D_1|_{\mathit{Locals}} \oplus D_2|_{\mathit{Globals}} & \text{otherwise} \end{cases}$$

560

In order to determine the effect of a call edge $k = (u, f();, v)$ we require abstract functions:

$$\begin{array}{lll} \text{enter}^\sharp & : & \mathbb{D} \to \mathbb{D} \\ \text{combine}^\sharp & : & \mathbb{D}^2 \to \mathbb{D} \end{array}$$

Then we define:

$$[\![k]\!]^\sharp \, D \quad = \quad \text{combine}^\sharp \, (D, [\![f]\!]^\sharp \, (\text{enter}^\sharp \, D))$$

Problems:

- How can we represent functions $f \;:\; \mathbb{D} \to \mathbb{D}$???
- If $\#\mathbb{D} = \infty$, then $\mathbb{D} \to \mathbb{D}$ has infinite strictly increasing chains :-(

Simplification: Copy-Constants

→ Conditions are interpreted as $;$:-)

→ Only assignments $x = e;$ with $e \in \mathit{Vars} \cup \mathbb{Z}$ are treated exactly :-)

Observation:

→ The effects of assignments are:

$$[\![x = e;]\!]^\sharp \, D \quad = \quad \begin{cases} D \oplus \{x \mapsto c\} & \text{if} \quad e = c \in \mathbb{Z} \\ D \oplus \{x \mapsto (D\ y)\} & \text{if} \quad e = y \in \mathit{Vars} \\ D \oplus \{x \mapsto \top\} & \text{otherwise} \end{cases}$$

→ Let $\mathbb{V}$ denote the (finite !!!) set of constant right-hand sides. Then variables may only take values from $\mathbb{V}^\top$:-))

→ The occurring effects can be taken from

$$\mathbb{D}_f \to \mathbb{D}_f \qquad \text{with} \qquad \mathbb{D}_f = (\mathit{Vars} \to \mathbb{V}^\top)_\perp$$

→ The complete lattice is huge, but finite !!!

Improvement:

→ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur :-)

→ All occurring functions $\lambda D.\bot \neq M$ are of the form:

$$\begin{array}{lll} M & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y \in I_x} y) \mid x \in \mathit{Vars}\} \qquad \text{where:} \\ M\ D & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y \in I_x} D\ y) \mid x \in \mathit{Vars}\} \qquad \text{für} \quad D \neq \bot \end{array}$$

→ Let $\mathbb{M}$ denote the set of all these functions. Then for $M_1, M_2 \in \mathbb{M}$ $(M_1 \neq \lambda\, D.\, \bot \neq M_2)$:

$$(M_1 \sqcup M_2)\ x \quad = \quad (M_1\ x) \sqcup (M_2\ x)$$

→ For $k = \#\mathit{Vars}$, $\mathbb{M}$ has height $\mathcal{O}(k^2)$:-)

Problems:

- How can we represent functions $f : \mathbb{D} \to \mathbb{D}$???
- If $\#\mathbb{D} = \infty$, then $\mathbb{D} \to \mathbb{D}$ has infinite strictly increasing chains :-(

Simplification: Copy-Constants

→ Conditions are interpreted as $;$:-)

→ Only assignments $x = e;$ with $e \in \mathit{Vars} \cup \mathbb{Z}$ are treated exactly :-)

The effects $[\![f]\!]^\sharp$ then can be determined by a system of constraints over the complete lattice $\mathbb{D} \to \mathbb{D}$:

$[\![v]\!]^\sharp$	$\sqsupseteq$	Id	v	entry point
$[\![v]\!]^\sharp$	$\sqsupseteq$	$[\![k]\!]^\sharp \circ [\![u]\!]^\sharp$	$k = (u, \_, v)$	edge
$[\![f]\!]^\sharp$	$\sqsupseteq$	$[\![stop_f]\!]^\sharp$	$stop_f$	end point of f

$[\![v]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ describes the effect of all prefixes of computation forests w of a procedure which lead from the entry point to v :-)

(1) Functional Approach:

Let $\mathbb{D}$ denote a complete lattice of (abstract) states.

Idea:

Represent the effect of $f()$ by a function:

$$[\![f]\!]^\sharp : \mathbb{D} \to \mathbb{D}$$

Improvement:

→ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur :-)

→ All occurring functions $\lambda D. \bot \neq M$ are of the form:

$$M \;=\; \{x \mapsto (b_x \sqcup \textstyle\bigsqcup_{y \in I_x} y) \mid x \in \mathit{Vars}\} \qquad \text{where:}$$

$$M\; D \;=\; \{x \mapsto (b_x \sqcup \textstyle\bigsqcup_{y \in I_x} D\; y) \mid x \in \mathit{Vars}\} \qquad \text{für} \quad D \neq \bot$$

→ Let $\mathbb{M}$ denote the set of all these functions. Then for $M_1, M_2 \in \mathbb{M}$ $(M_1 \neq \lambda\, D.\, \bot \neq M_2)$:

$$(M_1 \sqcup M_2)\; x \;=\; (M_1\; x) \sqcup (M_2\; x)$$

→ For $k = \#\mathit{Vars}$, $\mathbb{M}$ has height $\mathcal{O}(k^2)$:-)

Improvement:

→ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur :-)

→ All occurring functions $\lambda D.\bot \neq M$ are of the form:

$$\begin{array}{lcll} M & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} y) \mid x \in Vars\} & \text{where:} \\ M\;D & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} D\;y) \mid x \in Vars\} & \text{für} \quad D \neq \bot \end{array}$$

→ Let $\mathbb{M}$ denote the set of all these functions. Then for $M_1, M_2 \in \mathbb{M}$ $(M_1 \neq \lambda\, D.\,\bot \neq M_2)$:

$$(M_1 \sqcup M_2)\;x \quad = \quad (M_1\;x) \sqcup (M_2\;x)$$

→ For $k = \#Vars$, $\mathbb{M}$ has height $\mathcal{O}(k^2)$:-)

Improvement (Cont.):

→ Also, composition can be directly implemented:

$$\begin{array}{rcll} (M_1 \circ M_2)\;x & = & b' \sqcup \bigsqcup_{y\in I'} y & \text{with} \\ b' & = & b \sqcup \bigsqcup_{z\in I} b_z & \\ I' & = & \bigcup_{z\in I} I_z & \text{where} \\ M_1\;x & = & b \sqcup \bigsqcup_{y\in I} y & \\ M_2\;z & = & b_z \sqcup \bigsqcup_{y\in I_z} y & \end{array}$$

→ The effects of assignments then are:

$$[\![x = e;]\!]^\sharp \quad = \quad \begin{cases} \mathsf{Id}_{Vars} \oplus \{x \mapsto c\} & \text{if} \quad e = c \in \mathbb{Z} \\ \mathsf{Id}_{Vars} \oplus \{x \mapsto y\} & \text{if} \quad e = y \in Vars \\ \mathsf{Id}_{Vars} \oplus \{x \mapsto \top\} & \text{otherwise} \end{cases}$$

Improvement (Cont.):

→ Also, composition can be directly implemented:

$$\begin{array}{rcll} (M_1 \circ M_2)\;x & = & b' \sqcup \bigsqcup_{y\in I'} y & \text{with} \\ b' & = & b \sqcup \bigsqcup_{z\in I} b_z & \\ I' & = & \bigcup_{z\in I} I_z & \text{where} \\ M_1\;x & = & b \sqcup \bigsqcup_{y\in I} y & \\ M_2\;z & = & b_z \sqcup \bigsqcup_{y\in I_z} y & \end{array}$$

→ The effects of assignments then are:

$$[\![x = e;]\!]^\sharp \quad = \quad \begin{cases} \mathsf{Id}_{Vars} \oplus \{x \mapsto c\} & \text{if} \quad e = c \in \mathbb{Z} \\ \mathsf{Id}_{Vars} \oplus \{x \mapsto y\} & \text{if} \quad e = y \in Vars \\ \mathsf{Id}_{Vars} \oplus \{x \mapsto \top\} & \text{otherwise} \end{cases}$$

565

Improvement:

→ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur :-)

→ All occurring functions $\lambda D.\bot \neq M$ are of the form:

$$\begin{array}{lcll} M & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} y) \mid x \in Vars\} & \text{where:} \\ M\;D & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} D\;y) \mid x \in Vars\} & \text{für} \quad D \neq \bot \end{array}$$

→ Let $\mathbb{M}$ denote the set of all these functions. Then for $M_1, M_2 \in \mathbb{M}$ $(M_1 \neq \lambda\, D.\,\bot \neq M_2)$:

$$(M_1 \sqcup M_2)\;x \quad = \quad (M_1\;x) \sqcup (M_2\;x)$$

→ For $k = \#Vars$, $\mathbb{M}$ has height $\mathcal{O}(k^2)$:-)

564