

A Generic Model of Functional Programming With Dynamic Optimization

Martin Grabmüller

Fachgebiet Übersetzerbau und Programmiersprachen
Institut für Softwaretechnik und Theoretische Informatik
Fakultät IV – Elektrotechnik und Informatik
Technische Universität Berlin

TFP 2006, 19-21 April, 2006

Introduction to Virtual Machines and Dynamic Optimization

Formal Framework of Dynamic Optimization

Example: Feedback-Driven Inlining

Future Work and Conclusion

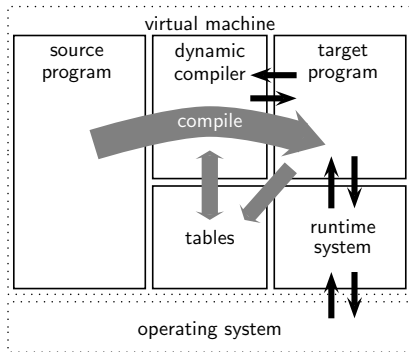
Introduction to Virtual Machines and Dynamic Optimization

Formal Framework of Dynamic Optimization

Example: Feedback-Driven Inlining

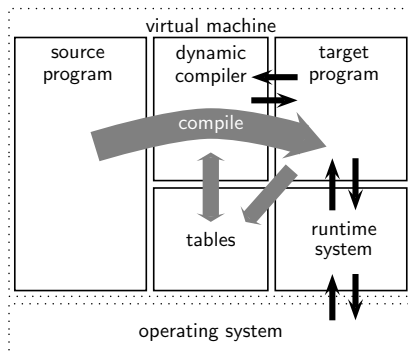
Future Work and Conclusion

Introduction to Virtual Machines



Virtual machine provides mapping

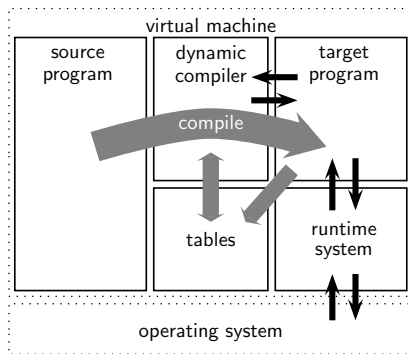
Introduction to Virtual Machines



Virtual machine provides mapping from

- ▶ expected instruction set to actual instruction set

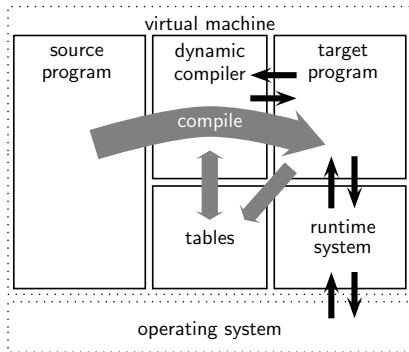
Introduction to Virtual Machines



Virtual machine provides mapping from

- ▶ expected instruction set to actual instruction set
- ▶ expected resources to actual resources

Introduction to Virtual Machines



Virtual machine provides mapping from

- ▶ expected instruction set to actual instruction set
- ▶ expected resources to actual resources

- ▶ Dynamic compilation translates *source* to *target* language at run time

Introduction to Dynamic Optimization

- ▶ Dynamic compilation translates *source* to *target* language at run time
- ▶ Dynamic optimization adds *analysis* and *optimizing transformation*

Introduction to Dynamic Optimization

- ▶ Dynamic compilation translates *source* to *target* language at run time
- ▶ Dynamic optimization adds *analysis* and *optimizing transformation*
- ▶ *Feedback-driven* (or *adaptive*) systems tailor translation and optimization to individual program runs

Introduction to Dynamic Optimization

- ▶ Dynamic compilation translates *source* to *target* language at run time
- ▶ Dynamic optimization adds *analysis* and *optimizing transformation*
- ▶ *Feedback-driven* (or *adaptive*) systems tailor translation and optimization to individual program runs
 - ▶ Transformation takes *input values* into account and is *lazy* (demand-driven)

Introduction to Dynamic Optimization

- ▶ Dynamic compilation translates *source* to *target* language at run time
- ▶ Dynamic optimization adds *analysis* and *optimizing transformation*
- ▶ *Feedback-driven* (or *adaptive*) systems tailor translation and optimization to individual program runs
 - ▶ Transformation takes *input values* into account and is *lazy* (demand-driven)
 - ▶ Several levels of optimization

Introduction to Virtual Machines and Dynamic Optimization

Formal Framework of Dynamic Optimization

Example: Feedback-Driven Inlining

Future Work and Conclusion

Formal framework is intended to cover several aspects of dynamic optimization

- ▶ Dynamic transformation

Formal framework is intended to cover several aspects of dynamic optimization

- ▶ Dynamic transformation
- ▶ Transformation on demand

Formal framework is intended to cover several aspects of dynamic optimization

- ▶ Dynamic transformation
- ▶ Transformation on demand
- ▶ Feedback-driven Optimization

Formal framework is intended to cover several aspects of dynamic optimization

- ▶ Dynamic transformation
- ▶ Transformation on demand
- ▶ Feedback-driven Optimization
- ▶ Support for local and global analysis and transformation

Source and Target Language

$P_s ::= \overline{D}_s E_s$	source program
$D_s ::= x = E_s$	source definition
$E_s ::= \textit{depends on source language}$	source expression
$P_t ::= \overline{D}_t E_t$	target program
$D_t ::= x = E_t$	target definition
$E_t ::= \textit{depends on target language}$	target expression
$\rightarrow : \overline{D}_t \times E_t \times K \rightarrow E_t \times K$	target reduction
$K ::= \textit{depends on source/target language}$	knowledge base
$\tau : \overline{D}_s \times E_s \times K \rightarrow E_t$	dynamic translation

$VM ::= (P_s, P_t, \rightarrow, \tau, \sigma, K)$ virtual machine

$\sigma : P_s \times P_t \times \tau \times K \rightarrow P_t \times K$ strategy function

- ▶ VM holds complete state of virtual machine

$VM ::= (P_s, P_t, \rightarrow, \tau, \sigma, K)$ virtual machine

$\sigma : P_s \times P_t \times \tau \times K \rightarrow P_t \times K$ strategy function

- ▶ VM holds complete state of virtual machine
- ▶ Strategy functions handles (re-)compilation using the transformation function

Program evaluation proceeds as follows:

1. Machine initialization:

Program evaluation proceeds as follows:

1. Machine initialization:
 - ▶ Load source program

Program evaluation proceeds as follows:

1. Machine initialization:

- ▶ Load source program
- ▶ Translate main program using transformation function τ

Program evaluation proceeds as follows:

1. Machine initialization:
 - ▶ Load source program
 - ▶ Translate main program using transformation function τ
2. Execution: Repeat in alternation:

Program evaluation proceeds as follows:

1. Machine initialization:

- ▶ Load source program
- ▶ Translate main program using transformation function τ

2. Execution: Repeat in alternation:

- ▶ Strategy function determines whether (re-)compilation is needed

Program evaluation proceeds as follows:

1. Machine initialization:

- ▶ Load source program
- ▶ Translate main program using transformation function τ

2. Execution: Repeat in alternation:

- ▶ Strategy function determines whether (re-)compilation is needed
- ▶ Reduction function reduces target program state

Program evaluation proceeds as follows:

1. Machine initialization:
 - ▶ Load source program
 - ▶ Translate main program using transformation function τ
2. Execution: Repeat in alternation:
 - ▶ Strategy function determines whether (re-)compilation is needed
 - ▶ Reduction function reduces target program state
3. When no changes occur anymore, the machine halts.

Introduction to Virtual Machines and Dynamic Optimization

Formal Framework of Dynamic Optimization

Example: Feedback-Driven Inlining

Future Work and Conclusion

- ▶ Profiling counts function applications

- ▶ Profiling counts function applications
- ▶ When a certain threshold is reached, body of called function is substituted for application expression

Feedback-Driven Inlining

- ▶ Profiling counts function applications
- ▶ When a certain threshold is reached, body of called function is substituted for application expression
- ▶ Inlining is tailored to each program run

Source and Target Language

$$E_s ::= x \mid \lambda x. E_s \mid E_s^l E_s \mid \text{zero} \mid \text{succ } E_s \mid \text{pred } E_s \mid \text{if0 } E_s E_s E_s$$
$$E_t \stackrel{\text{def}}{=} E_s$$
$$C_s ::= \bullet \mid \bullet E_s \mid v \bullet \mid \text{succ } \bullet \mid \text{pred } \bullet \mid \text{if0 } \bullet E_s E_s$$
$$C_t \stackrel{\text{def}}{=} C_s$$

where $v ::= \text{zero} \mid \text{succ } v \mid \lambda x. E_s$

Source and Target Language

$$E_s ::= x \mid \lambda x. E_s \mid E_s^l E_s \mid \text{zero} \mid \text{succ } E_s \mid \text{pred } E_s \mid \text{if0 } E_s E_s E_s$$
$$E_t \stackrel{\text{def}}{=} E_s$$

$$C_s ::= \bullet \mid \bullet E_s \mid v \bullet \mid \text{succ } \bullet \mid \text{pred } \bullet \mid \text{if0 } \bullet E_s E_s$$
$$C_t \stackrel{\text{def}}{=} C_s$$

where $v ::= \text{zero} \mid \text{succ } v \mid \lambda x. E_s$

$$K ::= \cdot \mid (l, n), K$$
$$K \oplus l = (l, 1), K \quad \text{if } l \notin K$$
$$K \oplus l = K_1, (l, n+1), K_2 \quad \text{if } K = K_1, (l, n), K_2$$
$$K \downarrow l = 0 \quad \text{if } l \notin K$$
$$K \downarrow l = n \quad \text{if } (l, n) \in K$$

Reduction Function

$$(\overline{D}_t, x, K) \rightarrow (\overline{D}_t \downarrow x, K) \quad (r1)$$

$$(\overline{D}_t, \text{if0 zero } E_{t1} E_{t2}, K) \rightarrow (E_{t1}, K) \quad (r2)$$

$$(\overline{D}_t, \text{if0 (succ } v) E_{t1} E_{t2}, K) \rightarrow (E_{t2}, K) \quad (r3)$$

$$(\overline{D}_t, \text{pred(succ } v), K) \rightarrow (v, K) \quad (r4)$$

$$(\overline{D}_t, (\lambda x. E_{t1})^l v_2, K) \rightarrow (E_{t1}[v_2/x], K \oplus l) \quad (r5)$$

$$(\overline{D}_t, E, K) \rightarrow (E', K') \Rightarrow (\overline{D}_t, C_t[E], K) \rightarrow (C_t[E'], K')$$

$$\sigma((\overline{D}_s, E_s), (\overline{D}_t, C[x]), \tau, K) = \begin{matrix} (\overline{D}_t \oplus (x = \tau \overline{D}_s \llbracket \overline{D}_s \downarrow x \rrbracket K), K) & \text{if } x \notin \overline{D}_t \end{matrix} \quad (s1)$$

$$\sigma((\overline{D}_s, E_s), (\overline{D}_t, C[(\lambda x. E_2) v_2]), \tau, K) = \begin{matrix} (\overline{D}_t \oplus (x = \tau \overline{D}_s \llbracket \overline{D}_s \downarrow x \rrbracket K), K) \\ \text{where } x \text{ contains } l \text{ and } K \downarrow l \geq T \end{matrix} \quad (s2)$$

Transformation Rules

$$\begin{aligned}\tau \overline{D_s} \llbracket x \rrbracket K &= x \\ \tau \overline{D_s} \llbracket \text{zero} \rrbracket K &= \text{zero} \\ \tau \overline{D_s} \llbracket \lambda x. E_s \rrbracket K &= \lambda x. \tau \overline{D_s} \llbracket E_s \rrbracket K \\ \tau \overline{D_s} \llbracket x^l E_{s2} \rrbracket K &= (\tau \overline{D_s} \llbracket x \rrbracket K)^l (\tau \overline{D_s} \llbracket E_{s2} \rrbracket K) \\ &\quad \text{if } K \downarrow l < T \\ \tau \overline{D_s} \llbracket x^l E_{s2} \rrbracket K &= \tau' \llbracket (\tau \overline{D_s} \llbracket \overline{D_s} \downarrow x \rrbracket K)^l \\ &\quad (\tau \overline{D_s} \llbracket E_{s2} \rrbracket K) \rrbracket \text{ if } K \downarrow l \geq T \\ \tau \overline{D_s} \llbracket E_{s1}^l E_{s2} \rrbracket K &= (\tau \overline{D_s} \llbracket E_{s1} \rrbracket K)^l (\tau \overline{D_s} \llbracket E_{s2} \rrbracket K) \\ \tau \overline{D_s} \llbracket \text{succ } E_s \rrbracket K &= \text{succ } (\tau \overline{D_s} \llbracket E_s \rrbracket K) \\ \tau \overline{D_s} \llbracket \text{pred } E_s \rrbracket K &= \text{pred } (\tau \overline{D_s} \llbracket E_s \rrbracket K) \\ \tau \overline{D_s} \llbracket \text{if0 } E_{s1} E_{s2} E_{s3} \rrbracket K &= \text{if0 } (\tau \overline{D_s} \llbracket E_{s1} \rrbracket K) (\tau \overline{D_s} \llbracket E_{s2} \rrbracket K) \\ &\quad (\tau \overline{D_s} \llbracket E_{s3} \rrbracket K) \\ \tau' : E_t &\rightarrow E_t \\ \tau' \llbracket (\lambda x. E_{t1})^l y \rrbracket &= E_{t1}[y/x] \\ \tau' \llbracket (\lambda x. E_{t1})^l \text{zero} \rrbracket &= E_{t1}[\text{zero}/x] \\ \tau' \llbracket E \rrbracket &= E \quad \text{if } E \text{ is complex}\end{aligned}$$

Example Program

$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$
--

$g = \lambda c. \text{pred } c$

$f^{/3} (\text{succ } (\text{succ } \text{zero}))$
--

$f^{/3} (\text{succ } (\text{succ } \text{zero}))$
--

.

Example Program

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$$
$$g = \lambda c. \text{pred } c$$

$$f^{/3} (\text{succ } (\text{succ } \text{zero}))$$

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$$
$$f^{/3} (\text{succ } (\text{succ } \text{zero}))$$

$$\cdot$$

Example Program

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$$
$$g = \lambda c. \text{pred } c$$

$$f^{/3} (\text{succ } (\text{succ } \text{zero}))$$

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$$
$$g = \lambda c. \text{pred } c$$

$$\text{zero}$$

$$/1 = 1, /2 = 2, /3 = 1, \cdot$$

Example Program

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (g^{/2} n))$$
$$g = \lambda c. \text{pred } c$$

$$f^{/3} (\text{succ } (\text{succ } \text{zero}))$$

$$f = \lambda n. \text{if0 } n \text{ zero } (f^{/1} (\text{pred } n))$$
$$g = \lambda c. \text{pred } c$$

$$\text{zero}$$

$$/1 = 1, /2 = 2, /3 = 1, \cdot$$

Introduction to Virtual Machines and Dynamic Optimization

Formal Framework of Dynamic Optimization

Example: Feedback-Driven Inlining

Future Work and Conclusion

Future work in the following areas is planned:

- ▶ Dynamic code loading

Future work in the following areas is planned:

- ▶ Dynamic code loading
- ▶ Generalize common aspects of languages and integrate them into basic model

Future work in the following areas is planned:

- ▶ Dynamic code loading
- ▶ Generalize common aspects of languages and integrate them into basic model
- ▶ Examine further optimizations

Future work in the following areas is planned:

- ▶ Dynamic code loading
- ▶ Generalize common aspects of languages and integrate them into basic model
- ▶ Examine further optimizations
- ▶ Study effects on language properties

Future work in the following areas is planned:

- ▶ Dynamic code loading
- ▶ Generalize common aspects of languages and integrate them into basic model
- ▶ Examine further optimizations
- ▶ Study effects on language properties
- ▶ Prototypical implementation to study optimization effects

We have seen...

- ▶ ... a simple generic theoretical model of functional virtual machines with
 - ▶ language independence, and
 - ▶ separation of concerns of program execution, analysis and transformation.

We have seen...

- ▶ ... a simple generic theoretical model of functional virtual machines with
 - ▶ language independence, and
 - ▶ separation of concerns of program execution, analysis and transformation.
- ▶ ... how to transfer a common optimization (feedback-driven inlining) into our model.

We have seen...

- ▶ ... a simple generic theoretical model of functional virtual machines with
 - ▶ language independence, and
 - ▶ separation of concerns of program execution, analysis and transformation.
- ▶ ... how to transfer a common optimization (feedback-driven inlining) into our model.

Thank You!