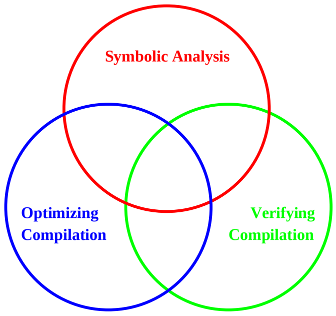
Symbolic Analysis in Optimizing and Verifying Compilation: Opportunities, Perspectives, Challenges

Jens Knoop

Technische Universität Wien, Austria

Technische Universität Wien
Vienna University of Technology

Motivation



Why Considering Code Motion?

...because it is

- Relevant ...widely used in practice
- General ...a family of optimizations rather than a single one
- Well-understood ...manually proven correct and optimal
- Challenging ...conceptually simple, but exhibits lots of thought-provoking phenomena

Last but not least...

- Regarding symbolic analysis ...allows conclusions relevant also beyond code motion

Structure of the Talk

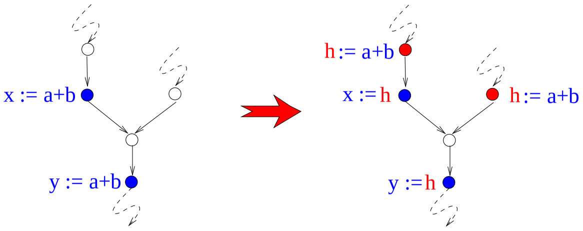
Exploring the design space...

- Part I: Code motion – The basics
- Part II: Code motion – In more detail
- Part III: Opportunities, Perspectives, Challenges
- Conclusions

Part I: Code Motion – What's it all about?

Traditionally...

- code motion (CM) means partial redundancy elimination (PRE)



Conceptually

...code motion can be considered a two-stage process

1. Expression hoisting

   ...hoisting expressions to "earlier" safe computation points

2. Total redundancy elimination

   ...eliminating computations which became totally redundant

Correctness, Optimality

One can prove...

- Correctness

  Theorem ["Essence"]

  ...at every use site of a temporary, recomputing the expression yields the same value which is stored in the temporary

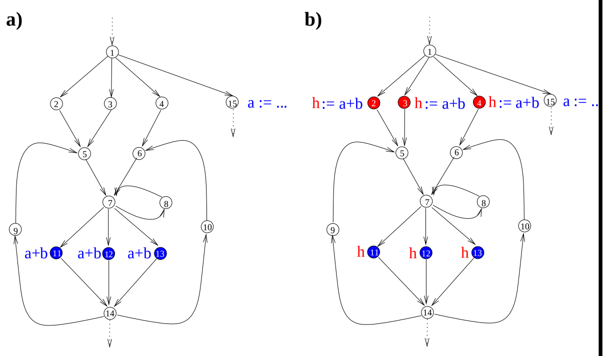
- Optimality

  Theorem [Earliestness principle]

  ...hoisting expressions to their earliest safe computation points leads to computationally optimal programs

  $\rightsquigarrow$ ...known as Busy Code Motion (PLDI'92, Knoop et al.)

A More Complex Example



Note

Traditionally,

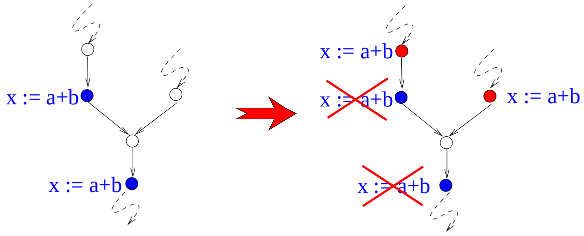
- Code (C) means expressions
- Motion (M) means hoisting

But...

- CM is more than hoisting of expressions and PR(E)E!

For example, code...

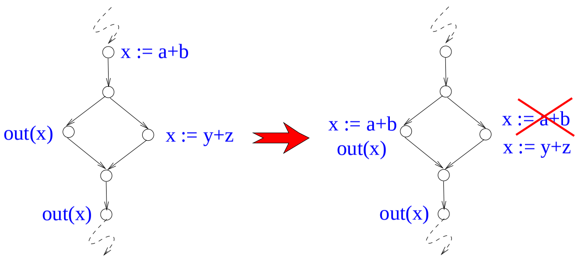
...can be assignments, too.



- Here, CM means partially redundant assignment elimination (PRAE)

In contrast to expressions, assignments...

...might also be sunk.



- Now, CM means partially dead code elimination (PDCE)

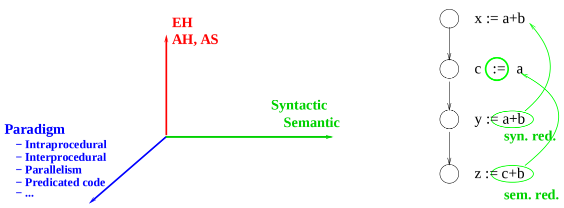
Towards the Design Space of CM-Algorithms...

More generally...

- Code means expressions/assignments
- Motion means hoisting/sinking

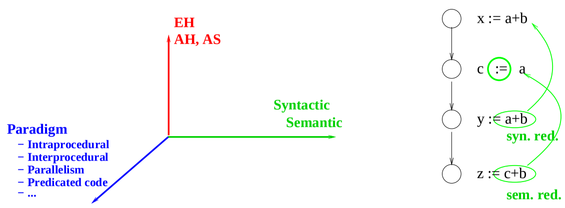
Code / Motion	Hoisting	Sinking
Expressions	EH	·/·
Assignments	AH	AS

Refining the Design Space of CM-Algorithms...



Introducing semantics... !

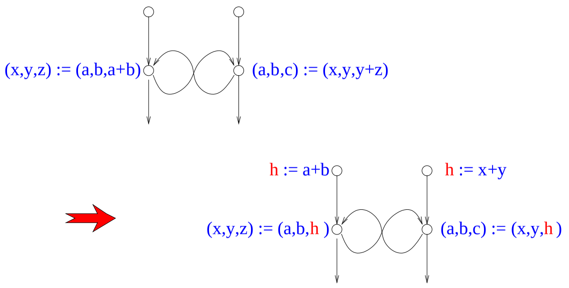
Refining the Design Space of CM-Algorithms...



Introducing semantics... !

Semantic Code Motion...

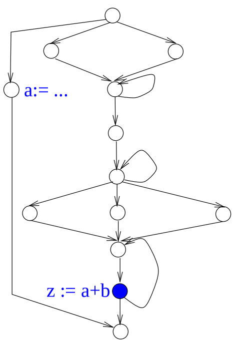
allows more powerful optimizations!



(example by B. Steffen, TAPSOFT'87)

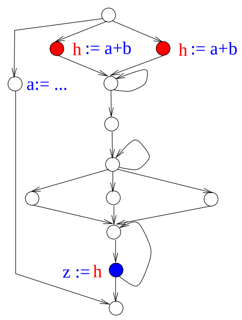
Code Motion (PREE)

...and its optimization goals!



Earliestness Principle

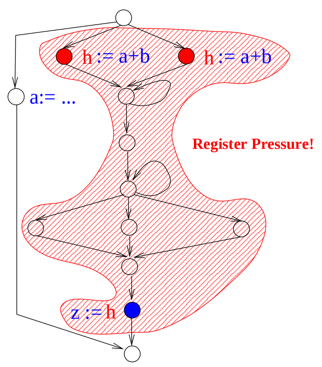
...yields computationally optimal programs.



⤳ Speed

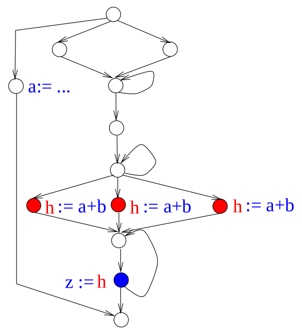
Earliestness Principle

...yields computationally optimal programs.



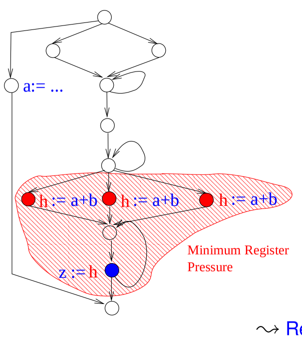
Latestness Principle

...yields computationally & lifetime optimal programs



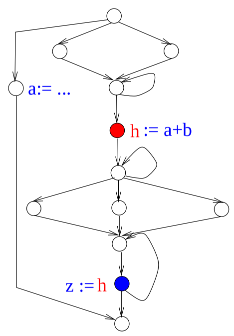
Latestness Principle

...yields computationally & lifetime optimal programs



⤳ Register pressure

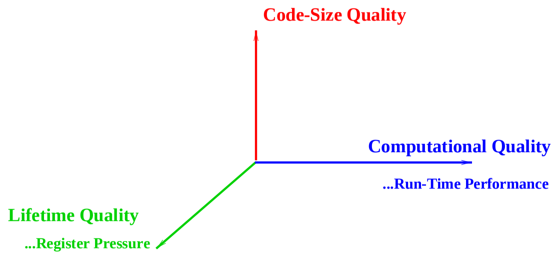
A Computationally and Code-Size Optimal Program



⤳ Code size

Part II: Code Motion in More Detail

...enhancing PRE to allowing a prioritization of goals



1999 World Market for Microprocessors

Going for size makes sense...

Chip Category	Number Sold
Embedded 4-bit	2000 million
Embedded 8-bit	4700 million
Embedded 16-bit	700 million
Embedded 32-bit	400 million
DSP	600 million
Desktop 32/64-bit	150 million

∼ 2%

... David Tennenhouse (Intel Director of Research). Keynote Speech at the *20th IEEE Real-Time Systems Symposium* (*RTSS'99*), Phoenix Arizona, December 1999.

Think of...

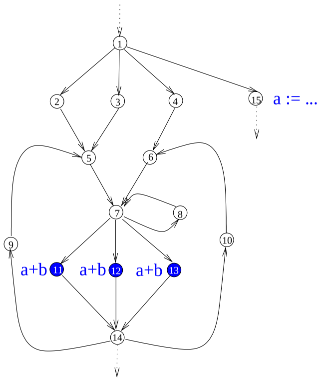
... domain-specific processors as used in embedded systems

- **Telecom**
  - Cell phones, pagers, ...
- **Consumer Electronics**
  - MP3 player, cameras, pocket games, ...
- **Automotive**
  - GPS navigation, airbags, ...
- ...

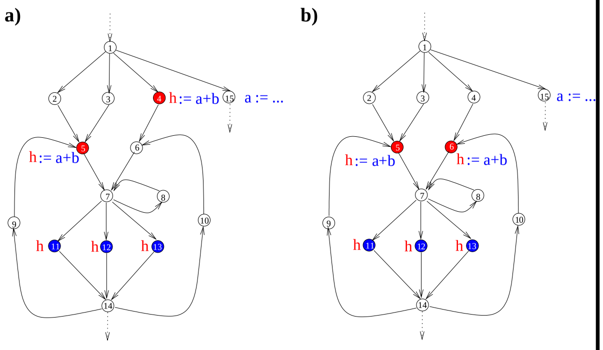
For such applications...

...code size often more critical than speed!

Going for Code-Size Sensitivity – Running Example



The Running Example (Cont'd)



SQ > CQ > LQ SQ > LQ > CQ

Background: Classical PRE

⇝ **Busy Code Motion (BCM)** (Knoop et al., PLDI'92)

...yields **computationally optimal** results!

Conceptually

- ...based on the **Earliestness Principle**,
  i.e., placing computations as early as possible,
  while preserving admissibility (semantics & performance)

Technically

...relies on just two GEN/KILL-Analyses

- UpSafety (also known as Availability)
- DownSafety (also known as Very Busyness)

Background: Classical PRE (Cont'd)

⇝ **Lazy Code Motion (LCM)** (Knoop et al., PLDI'92)

...yields **computationally & lifetime optimal** results!

Conceptually

- ...based on the **Latestness Principle**,
  i.e., placing computations as late as possible,
  while preserving admissibility&computational optimality

Technically

...requires in addition to the two analyses of **BCM** only one further GEN/KILL-Analysis

- Delayability

Code-Size Sensitive PRE

⇝ **The Problem**

...how to get an admissible code-size minimal placement of computations, i.e., a placement which is

- admissible (semantics & performance preserving)
- code-size minimal

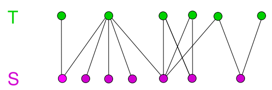
⇝ **The Idea** ... essentially, a trade-off problem

...trading the original computations against newly inserted ones!

⇝ **The Solution** ... **Graph Theory!**

...reducing the trade-off problem to the computation of tight sets in bipartite graphs based on maximum matchings!

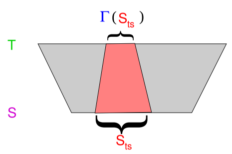
Bipartite Graph



Tight Set

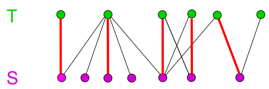
... of a bipartite graph $(S \cup T, E)$ is a subset $S_{ts} \subseteq S$ such that

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \ \geq\ |S'| - |\Gamma(S')|$$



Two Variants: (1) **Largest** Tight Sets (2) **Smallest** Tight Sets

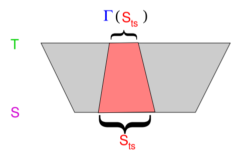
Bipartite Graph



Tight Set

... of a bipartite graph $(S \cup T, E)$ is a subset $S_{ts} \subseteq S$ such that

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \ \geq\ |S'| - |\Gamma(S')|$$



Two Variants: (1) **Largest** Tight Sets (2) **Smallest** Tight Sets

Computation of...

- **Maximum matchings** and
- **Tight sets**

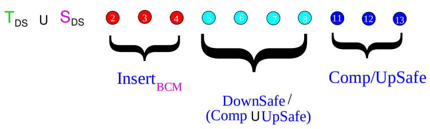
...use standard algorithms offered by graph theory!

Hence, the problem boils down to...

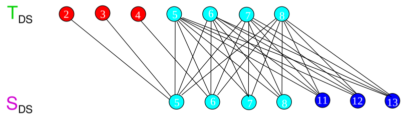
...constructing the bipartite graph modelling the problem!

Modelling the Trade-Off Problem

The Set of Nodes



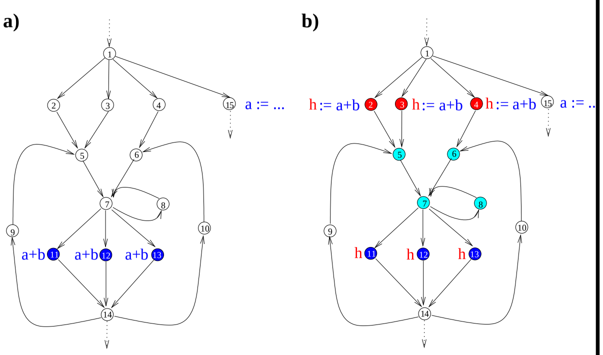
The Bipartite Graph



The Set of Edges ... $\forall\, n \in S_{DS}\ \forall\, m \in T_{DS}.$

$$\{n,m\} \in E_{DS} \iff_{df} m \in \textbf{Closure}(pred(n))$$

Remember...



DownSafety Closures

DownSafety Closure

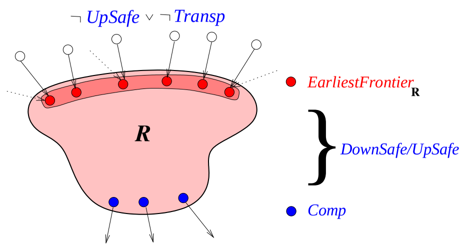
For $n \in$ *DownSafe/Upsafe* the DownSafety Closure $Closure(n)$ is the smallest set of nodes satisfying

1. $n \in Closure(n)$
2. $\forall\, m \in Closure(n) \setminus$ *Comp*. $succ(m) \subseteq Closure(n)$
3. $\forall\, m \in Closure(n).\ pred(m) \cap Closure(n) \neq \emptyset \Rightarrow$ $pred(m) \setminus$ *UpSafe* $\subseteq Closure(n)$

DownSafety Regions

A set $\mathcal{R} \subseteq N$ of nodes is a DownSafety Region if and only if

1. *Comp*$\setminus$*UpSafe* $\subseteq \mathcal{R} \subseteq$ *DownSafe*$\setminus$*UpSafe*
2. $Closure(\mathcal{R}) = \mathcal{R}$



Fundamental...

Characterization Theorem

Insertions of admissible PRE-Transformations are always at **"earliest-frontiers"** of **DownSafety regions**.

The Key Questions

1. Where to insert computations?
2. What is the impact on the code size?
3. Why is it optimal, i.e., code-size minimal?

...three theorems answering one of these questions each.

Main Results / First Question

1. Where to insert computations?

Theorem 1 [Tight Sets]

Let $TS \subseteq S_{DS}$ be a tight set.

Then $\mathcal{R}_{TS} =_{df} \Gamma(TS) \cup$ (*Comp*$\setminus$*UpSafe*)

is a DownSafety Region with *Body*$_{\mathcal{R}_{TS}} = TS$

Main Results / Second Question

2. What is the impact on the code size?

Theorem 2 [DownSafety Regions]

Let $\mathcal{R}$ be a DownSafety Region

with $Body_{\mathcal{R}} =_{df} \mathcal{R} \setminus EarliestFrontier_{\mathcal{R}}$

Then

- **Space Gain of Inserting at $EarliestFrontier_{\mathcal{R}}$:**

  $|Comp \setminus UpSafe| - |EarliestFrontier_{\mathcal{R}}| =$

  $|Body_{\mathcal{R}}| - |\Gamma(Body_{\mathcal{R}})| \;\; _{df}= defic(Body_{\mathcal{R}})$

Main Results / Third Question

3. Why is it optimal, i.e., code-size minimal?

Optimality Theorem [The Transformation]

Let $TS \subseteq S_{DS}$ be a tight set.

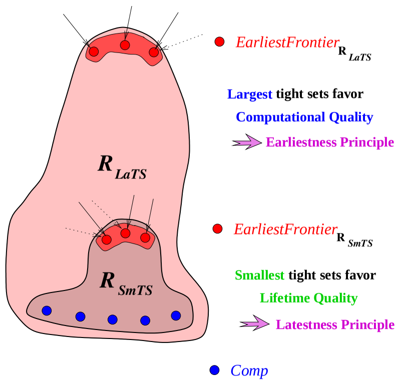
- **Insertion Points:**

  $Insert_{SpCM} =_{df} EarliestFrontier_{\mathcal{R}_{TS}} = \mathcal{R}_{TS} \setminus TS$

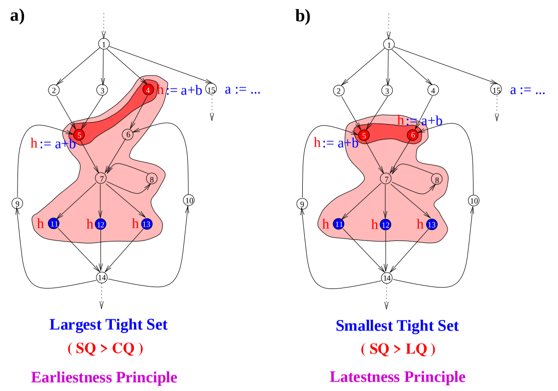
- **Space Gain:**

  $defic(TS) =_{df} |TS| - |\Gamma(TS)| \geq 0$ max.

Largest vs. Smallest Tight Sets: The Impact



The Running Example



Related Work ... A Brief History of PRE(E)

- $<$ **1979** ... Special Techniques

  ⇝ Total Redundancy Elimination, Loop Invariant Code Motion

- **1979** ... Partial Redundancy Elimination

  ⇝ **Pioneering** ... Morel/Renvoise's bidirectional algorithm [1979]

  ⇝ **Heuristic improvements** ... Dhamdhere [1988, 1991], Drechsler/Stadel [1988], Sorkin [1989], Dhamdhere/Rosen/Zadeck [1992], ...

- **1992** ... BCM & LCM [Knoop et al., PLDI'92]

  ⇝ BCM ... first to achieve Computational Optimality: Earliestness Principle

  ⇝ LCM ... first to achieve Comp. & Lifetime Optimality: Latestness Principle

  ... first to be purely unidirectional, however, not yet code-size sensitive.

- **2000**: Code-Size Sensitive PRE [Knoop et al., POPL 2000]

- **2004**: Speculative Code-Size Sensitive PRE [Knoop et al., LCTES 2004]

Code-Size Sensitive PRE at a Glance

Preprocess

- Optional: Perform LCM (3 GEN/KILL-DFAs)
- Compute Predicates of BCM for G resp. LCM (G) (2 GEN/KILL-DFAs)

Main Process

Reduction Phase

- Construct Bipartite Graph
- Compute Maximum Matching

Optimization Phase

- Compute Largest/Smallest Tight Set
- Determine Insertion Points

Choice of Priority	Apply	To	Using	Yields	Auxiliary Information Required
$\mathcal{LQ}$	Not meaningful: The identity, i.e., G itself is optimal!				
$\mathcal{SQ}$	Subsumed by $\mathcal{SQ} > \mathcal{CQ}$ and $\mathcal{SQ} > \mathcal{LQ}$!				
$\mathcal{CQ}$	BCM	G			UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{LQ}$	LCM	G		LCM(G)	UpSafe(G), DownSafe(G), Delay(G)
$\mathcal{SQ} > \mathcal{CQ}$	SpCM	G	Largest tight set	$\mathrm{SpCM}_{LTS}(\mathrm{G})$	UpSafe(G), DownSafe(G)
$\mathcal{SQ} > \mathcal{LQ}$	SpCM	G	Smallest tight set		UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{SQ}$	SpCM	LCM(G)	Largest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(**LCM**(G)), DownSafe(**LCM**(G))
$\mathcal{CQ} > \mathcal{SQ} > \mathcal{LQ}$	SpCM	LCM(G)	Smallest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(**LCM**(G)), DownSafe(**LCM**(G))
$\mathcal{SQ} > \mathcal{CQ} > \mathcal{LQ}$	SpCM	$\mathrm{DL}(\mathrm{SpCM}_{LTS}(\mathrm{G}))$	Smallest tight set		UpSafe(G), DownSafe(G), Delay($\mathbf{SpCM}_{LTS}$(G)), UpSafe(**DL**($\mathbf{SpCM}_{LTS}$(G))), DownSafe(**DL**($\mathbf{SpCM}_{LTS}$(G)))

Part III: Opportunities

Obvious hooks for symbolic analysis...

- "More" semantics, beyond Herbrand-semantics...
  ⇝ from interpreting := to interpreting operators (assoc., comm., distr.)
- "More" data structures...
  ⇝ from scalar variables to arrays, etc.

Obvious research problems...

- Do the correctness/optimality results remain valid?

Optimality Results are quite sensitive!

Three examples to give evidence...

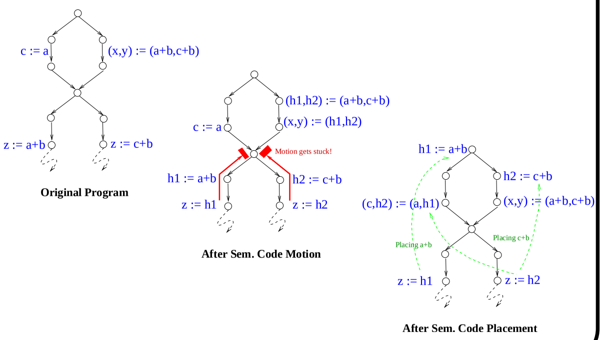
(I) Code motion vs. code placement

(II) Interdependencies of (elementary) transformations

(III) Paradigm dependencies

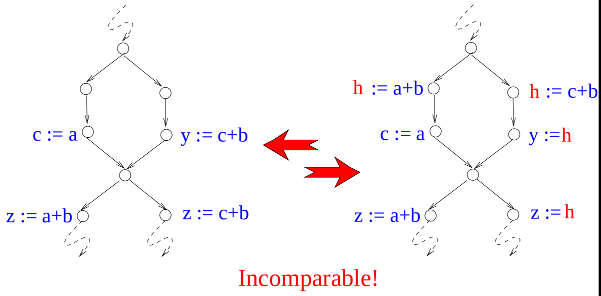
(I) Code Motion vs. Code Placement

...not just synonyms!



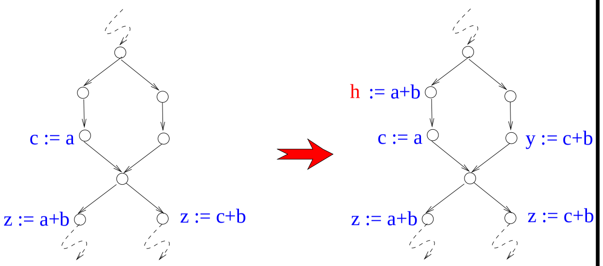
Even worse...

Optimality is lost!

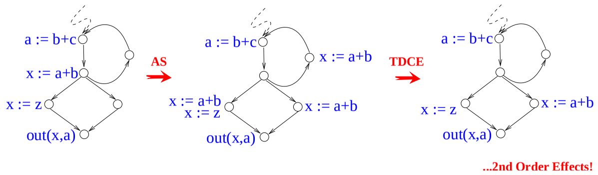


Even more worse...

Peformance may be lost, when naively applied!

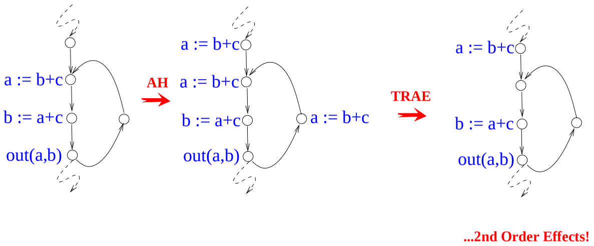


(II) Interdependencies of Transformations



⇝ ...Partial Dead-Code Elimination (PDCE)

Interdependencies of Transformations



⇝ ...Partially Redundant Assignment Elimination (PRAE)

Conceptually

...we can think of PREE, PRAE and PDCE in terms of

- PREE = AH ; TREE
- PRAE = (AH + TRAE)*
- PDCE = (AS + TDCE)*

PRAE/PDCE – Optimality Results

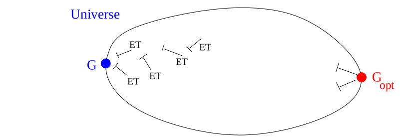
Derivation relation $\vdash$...

- PRAE... $G \vdash_{AH,TRAE} G'$ (ET={AH,TRAE})
- PDCE... $G \vdash_{AS,TDCE} G'$ (ET={AS,TDCE})

We can prove...

Optimality Theorem

For both PRAE and PDCE, $\vdash_{ET}$ is confluent and terminating



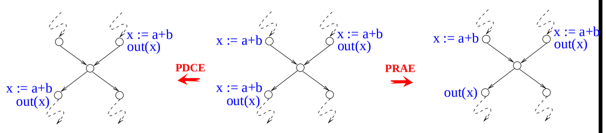
Consider now...

- Assignment Placement AP

  AP = (AH + TRAE + AS + TDCE)*

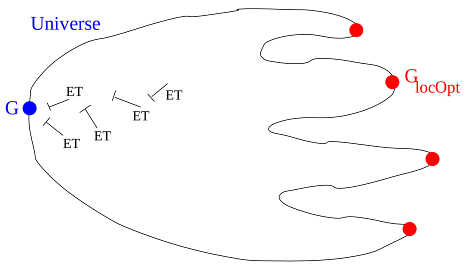
...should be even more powerful!

Indeed, but...



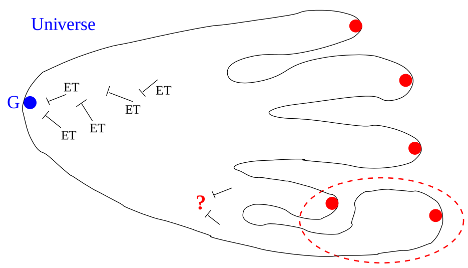
Confluence...

...and hence (global) optimality are lost!

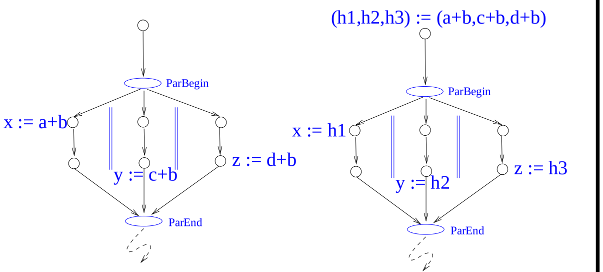


Even worse...

...there are scenarios, where we can end up with universes like



(III) Paradigm Dependencies



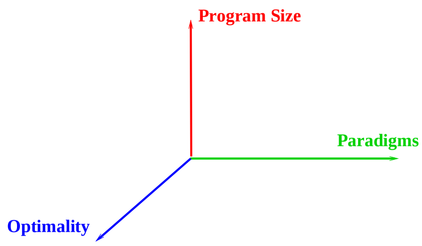
Original Program

After Earliestness Transformation

Challenges

...for symbolic analysis from the perspective of optimizing compilation:

Scalability!



Perspectives

...for symbolic analysis in compilation

- ...depend much on scalability
- ...less compile-time critical scenarios (+)
  - Compiling for embedded systems
  - Verifying compilation

    $\leadsto$ ...growing need for trustworthy/proven correct compilers

    cp. The Verifying Compiler: A Grand Challenge (Tony Hoare)

    ...especially challenging in this context ...optimizations
- ...more compile-time critical scenarios (–)
  - Just-in-time (JIT) compilation
  - Mobile code

Conclusions

Symbolic Analysis in Optimizing and Verifying Compilation...

- Challenges ...**scalability!**
- Opportunities ...numerous and obvious applications in optimizing compilation
  - Many research problems: e.g., which optimality results do hold?
- Perspectives
  - Verifying Compilation (+)
  - Compiling for Embedded Systems (+)
  - JIT Compilation (–)
  - Mobile Code (–)

Forthcoming Related Events

- 4th International Workshop on Compiler Optimization Meets Compiler Verification (COCV 2005) (Edinburgh, UK, April 3, 2005). In conjunction with ETAPS 2005.
  Submission deadline: **November 26, 2004**
  URL: www.complang.tuwien.ac.at/knoop/COCV2005/cocv2005.html
- Dagstuhl Seminar 05311 on "Verifying Optimizing Compilers", Schloss Dagstuhl, Germany, July 31 - August 5, 2005.
  URL: www.dagstuhl.de/05311/

Thank you!

Questions?

Acknowledgements: Most of the results reported are joint work with *Oliver Rüthing* (U. Dortmund), *Bernhard Steffen* (U. Dortmund), *Eduard Mehofer* (U. Wien), and more recently *Bernhard Scholz* (U. Sydney) and *Nigel Horspool* (U. Victoria).