AOStA

and some ideas

Marcus Denker
Software Composition Group
University of Berne

Overview

Part I:

AOStA Revisited: A Short Overview

Part II:

A Code Generator for AOStA

Part III:

Beyond Performance

Squeak3.7a-5526.1.image

Part I: AOStA

- AOStA: Adaptive Optimizations for Smalltalk

- Profiled execution to identify hot spots

- Compiles to optimized bytecode

- Dynamic deoptimization (debugging)

- Written in Smalltalk

More...

Profiled execution: two areas for JIT-compiled methods

- The optimized area works as usual
- In the unoptimized area methods have a counter for each send and backward branch

Collecting type information

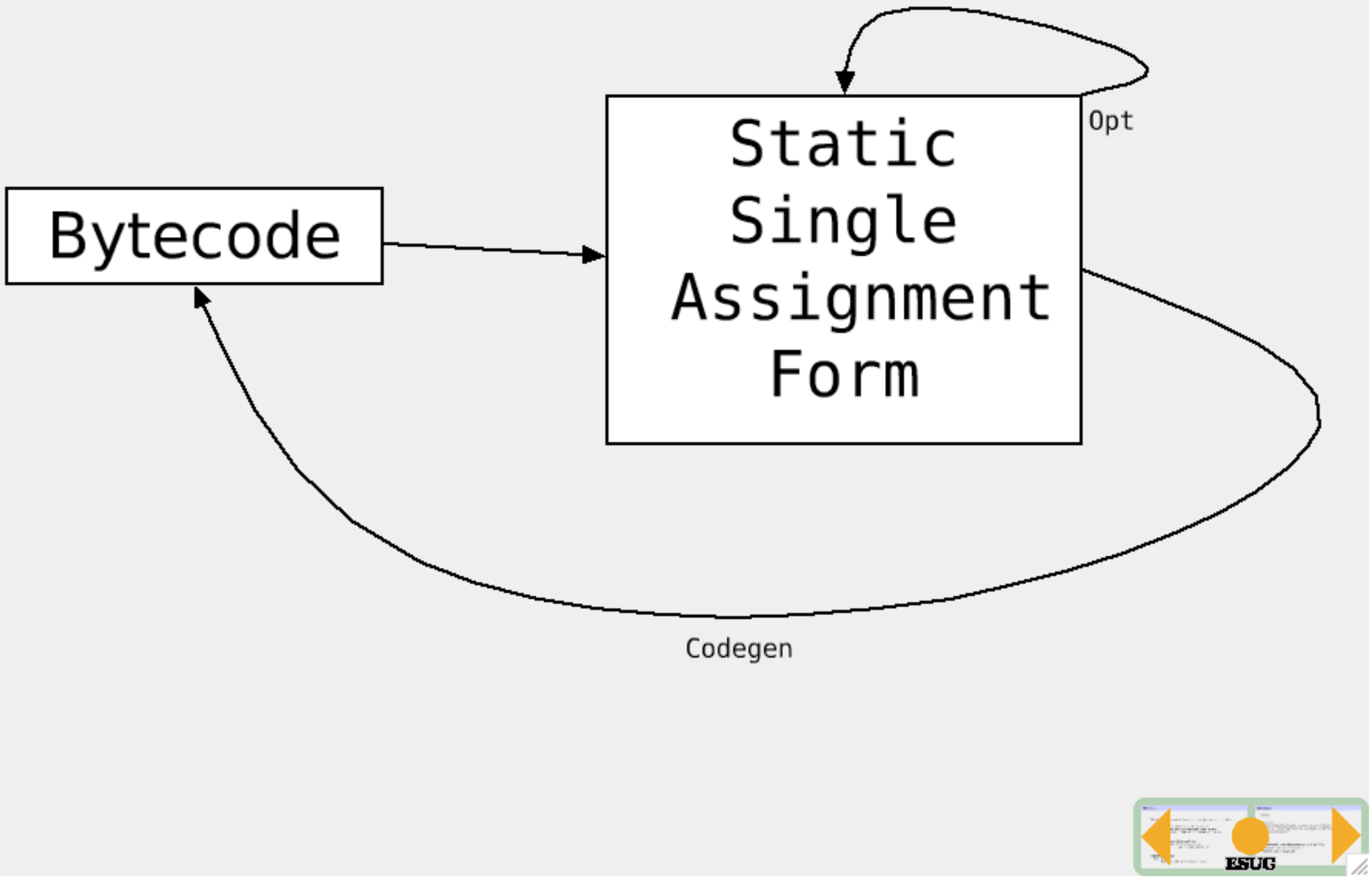
- via Polymorphic Inline Caches
- need to be readable from Smalltalk

Optimizations:

e.g., Inlining
Specializations for known types

Squeak3.7a-5526.1.image

Bytecode-to-Bytecode



Status

2003:

- Design
- Frontend: Bytecode transformed to SSA
- Middle: SSA Framework, sample optimizers
- For VisualWorks

2004:

- Backend: transformation out of SSA
- Simple Code Generator
- Done with Squeak

Squeak3.7a-5526.1.image

Part II: More about the backend

- Short introduction to SSA **(**Static Single Assigment)

- Two steps:

  1) Deconstruction of SSA

  2) Code generation

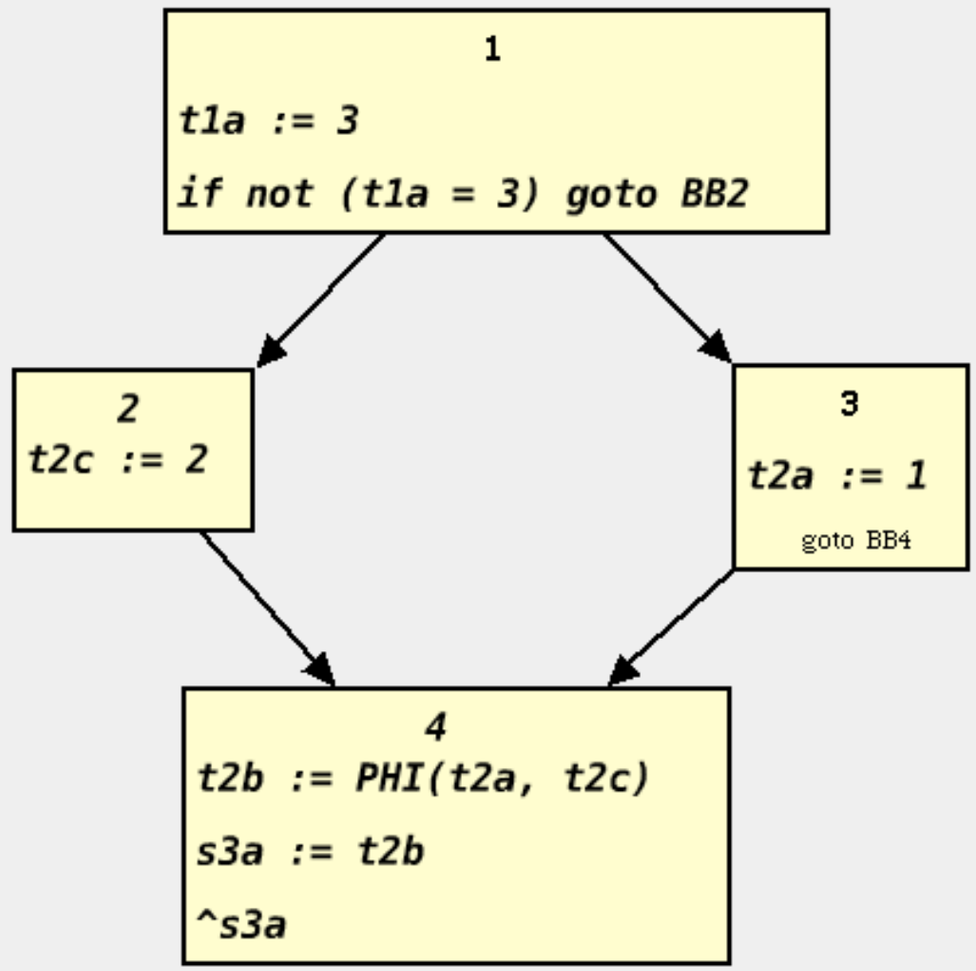
- Some examples

SSA - Static Single Assigment Form

- SSA: Each Variable has **one** assignment

- If controll flow merges, we need to select the variable from the path we came from

```
examplePhiSimple
   | a b |
   a := 3.
   a = 3
      ifTrue: [b := 1]
      ifFalse: [ b := 2 ].
   ^b
```



SSA

- Very nice for many optimizations

- but: Code generation not possible directly

--> Need to remove virtual selector functions
(PHI-functions)

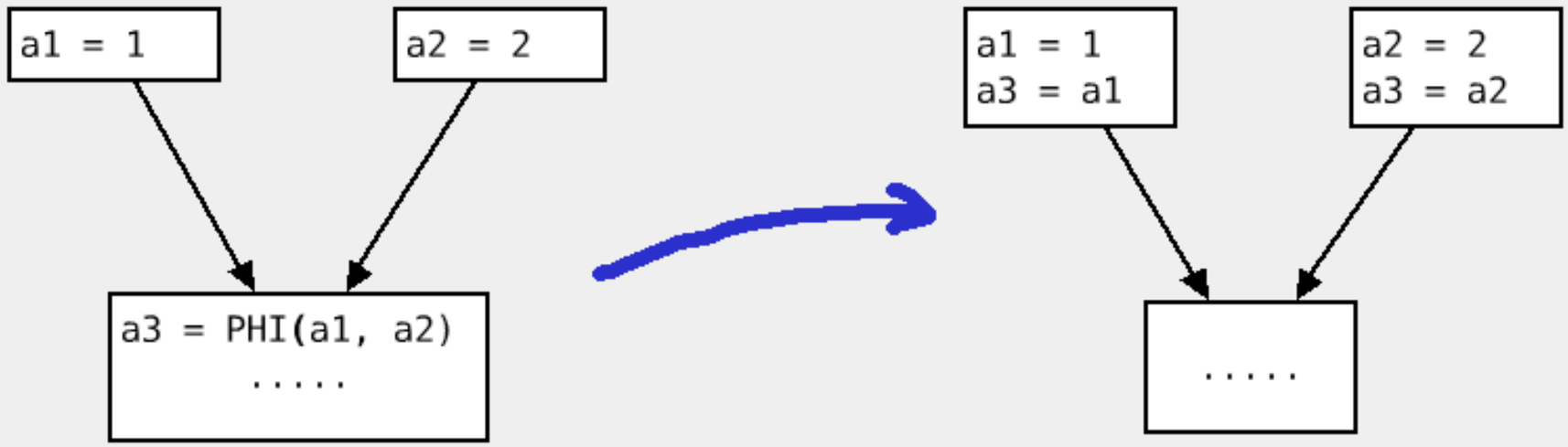
Two step code generation

1) Deconstruction of SSA

2) Code generation

SSA Deconstruction

Canonical method:



Problems:

- Wrong after some optimizations

- Copies need to be removed

Squeak3.7a-5526.1.image

Phi-Congruency Method

Method by Vugranam C. Sreedhar, Roy Dz-Ching Ju, David M. Gillies, und Vatsa Santhanam.

Idea: Transform program that all variables are the same in PHI:

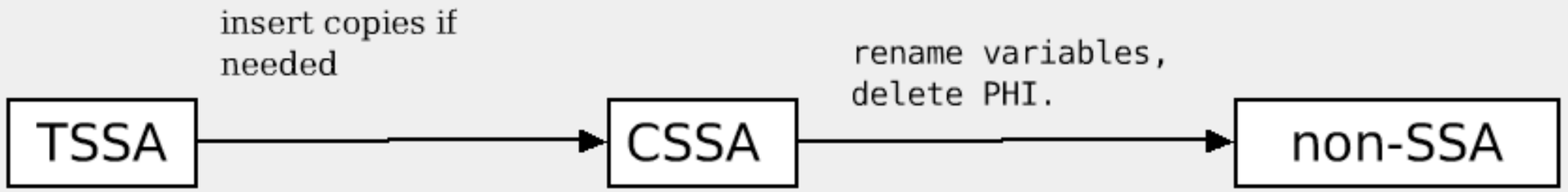
```
a1 = PHI(a1,a1)   ⟶   a1 = a1
```

- Insert copies

- Renaming

Phi-Congruency: Overview

- Two step process:



- Nice properties:
  - Without any optimizations, no copies are needed
  - Simple heuristics for copy placement

Number of copies

canonical: ~16000
new: ~1000 (without Opt: 0)

Squeak3.7a-5526.1.image

IRBuilder

Symbolic Assembler

Example:

```
| ir aCompiledMethod |
ir := IRBuilder new
        rargs: #(self);
        pushLiteral: 1;
        returnTop;
        ir.
aCompiledMethod := ir compiledMethod.
```

Execute:

```
aCompiledMethod valueWithReceiver: nil arguments: #()
```

Install in the system:

```
Float addSelector: #test withMethod: aCompiledMethod.
```

Squeak3.7a-5526.1.image

Example

```
examplePhiSimple
   | a b |
   a := 3.
   (a = 3)ifTrue: [b := 1]
          ifFalse: [b := 2].
   ^b
```

ClosureCompiler

CompiledMethod

AOStA

```
BB9: [
    t1a:= 3.
    if not (t1a = 3) goto BB18]
BB15: [
    t2a := 1.
    goto BB20]
BB18: [
    t2c := 2:]
BB20: [
    t2b := PHI(t2a, t2c).
    s3a := t2b.
    ^s3a]'
```

Opt and SSA
deconstruction

```
BB9: [
   t1a := 3.
   if not (t1a = 3:) goto
BB18]
BB15: [
   t3a := 1 ).
   goto BB20]
BB18: [
   t3a := 2]
BB20: [
   ^t3a]
```

Codegen

CompiledMethod

self	9 <20> pushConstant: 3
all bytecodes	10 <81 40> storeIntoTemp: 0
header	12 <20> pushConstant: 3
literal1	13 <B6> send: =
9	14 <9A> jumpFalse: 18
10	15 <76> pushConstant: 1
11	16 <69> popIntoTemp: 1
12	17 <91> jumpTo: 20
13	18 <77> pushConstant: 2
14	

```
self valueWithReceiver: nil
      arguments: #()
```

Part III: TODO/Ideas

TODO... lots. e.g. dynamic deoptimization

Possible experiments:

-> AOStA on Squeak with Jitter
-> Does it make sense with just an interpreter?
-> Exupery as a backend

All these are related to perfomance.

Question: What else could be possible?

Squeak3.7a-5526.1.image

Runtime Translation as a System Service

- Enables more late binding

  Example:

  - iVars are accessed via offsets
  - offsets are calculated at compile time
  - makes changes and experiments harder

- Make a MOP practical

- Allows a much simpler System

Squeak3.7a-5526.1.image

MOPs and other strange stuff

MOP: Meta Object Protocol.

Idea: Provide an API for changing the language semantics and implementation at runtime.

(e.g., meaning of inheritance)

For Squeak: MetaClassTalk

- Nice, but slow
- A runtime translator could regain performance

Example: ClassBoxes

Two Kinds of Bytecode

"Image"level Vs. Interpreter Level

- Imagelevel bytecode can be simple:
  => No optimizations at all

- Imagelevel bytecode and interpreter bytecode could even be different:
  => Latebinding of the execution format

- Why not just use the AST?

Squeak3.7a-5526.1.image

"2 Worlds"

Software-Engineering

- AST instead of Bytecode
- late bound
- no optimizations

Translator

Execution

- bytecode or binary
- optimized
- late binding resolved