

Notations in SolidOpt



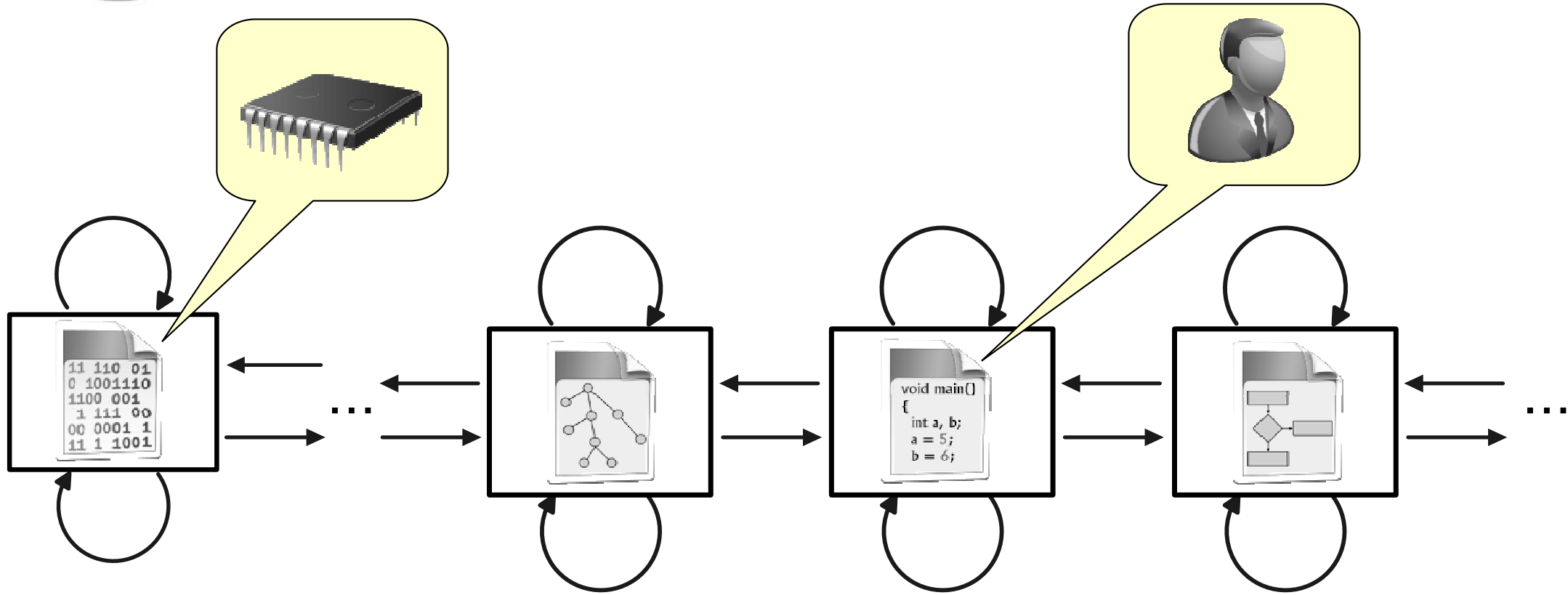
Vassil Vassilev

Contents

1. Introduction
2. Definitions
3. SolidOpt Notation Graph
4. Control Flow Graph
5. Call Graph
6. Three address code representation
7. Single Static Assignment Form
8. Conclusion



Introduction



Definitions



Algorithm

- N-tuple of executable commands
- Transition from a start state to an end state



Command/Step

- Action, which changes the state of the system



Notation

- A way of describing the algorithm



Executor

- !?

Definitions



Algorithm

- N-tuple of executable instructions
- Transition of an input to an output data



Command/Step

- Action, which changes the state of the computer system (memory, peripheral devices, etc)



Notation

- A way of description of the algorithm – machine code, different programming languages



Executor

- CPU

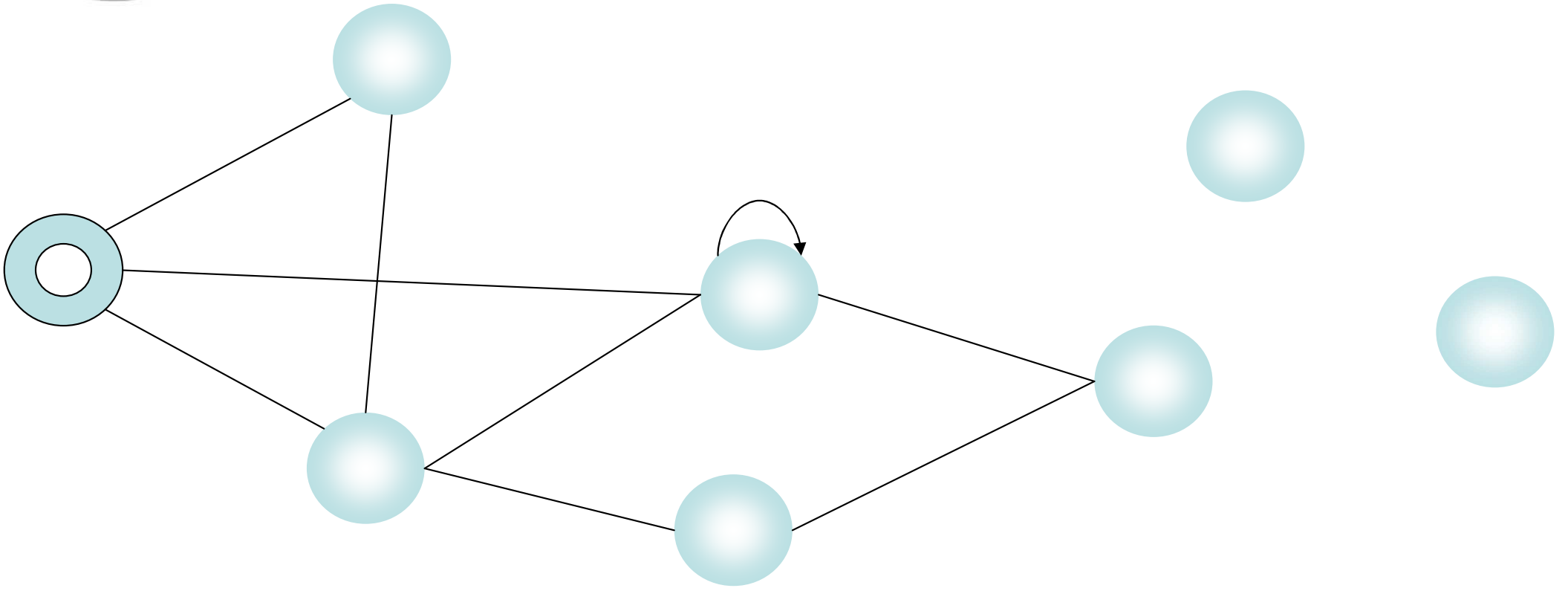


SolidOpt Notation Graph ?!

Multigraph consisting of transformation modules:

- ❖ Optimization modules
- ❖ Representations (Notations)
- ❖ Transitions between the representations

SolidOpt Notation Graph





Why we need multiple notations

- ❖ different executors can understand them
- ❖ allow different transformations
- ❖ pinpoint certain parts of the algorithm

Characteristics



Dynamic creation, depending on:

- ❖ loaded notations
- ❖ loaded transitions between the notations
- ❖ ...

Flow Graphs



Creation of graphs, reflecting the explicit change of the IP (Instruction Pointer)

- ❖ Branches (Control Flow Graphs)
- ❖ Subroutines (Call Graphs)
- ❖ ...

Advantages

- ❖ All the mathematical algorithms for graphs are valid
- ❖ Various types of deduction on their basis
- ❖ ...

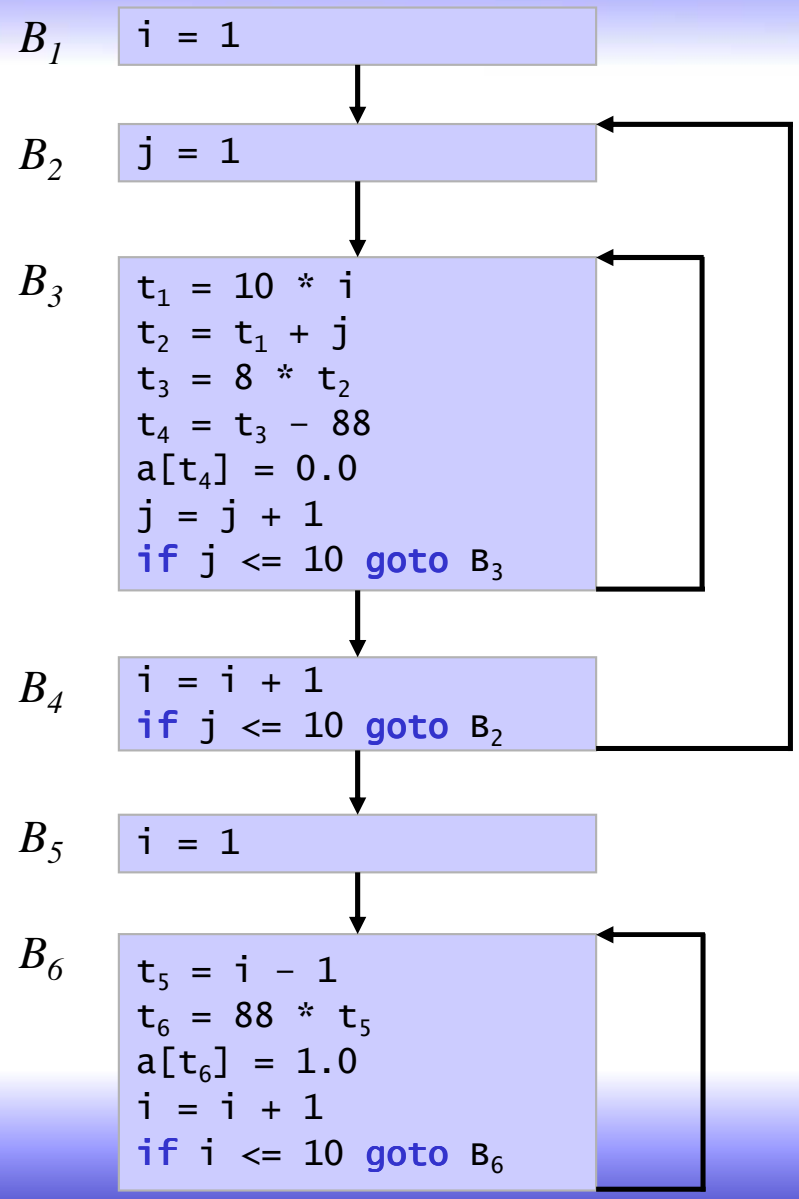


Control Flow Graph

Represent the algorithm as connected graph. The conditions for connectivity are:

- ❖ There is conditional or unconditional jump from the end of B to the beginning of C;
- ❖ C immediately follows B in the original order of instructions and B does not end in an unconditional jump.

```
// Turns matrix 10x10 into and identity matrix
for i from 1 to 10 do
  for j from 1 to 10 do
    a[i,j] = 0.0;
for i from 1 to 10 do
  a[i,i] = 1.0;
```



CFG in SolidOpt



B_1

```
IL_0000: ldc.i4.s 10
IL_0002: ldc.i4.s 10
IL_0004: newobj instance
IL_0009: stloc.0
IL_000a: ldc.i4.1
IL_000b: stloc.1
IL_000c: br.s IL_0030
```

B_2

```
IL_0030: ldloc.1
IL_0031: ldc.i4.s 10
IL_0033: ble.s IL_000e
```

B_7

```
IL_0035: ldc.i4.1
IL_0036: stloc.3
IL_0037: br.s IL_004e
```

B_3

```
IL_000e: ldc.i4.1
IL_000f: stloc.2
IL_0010: br.s IL_0027
```

B_8

```
IL_004e: ldloc.3
IL_004f: ldc.i4.s 10
IL_0051: ble.s IL_0039
```

B_{10}

```
IL_0053: ldc.i4.1
IL_0054: call ReadKey(boo1)
IL_0059: pop
IL_005a: ret
```

B_4

```
IL_0027: ldloc.2
IL_0028: ldc.i4.s 10
IL_002a: ble.s IL_0012
```

B_6

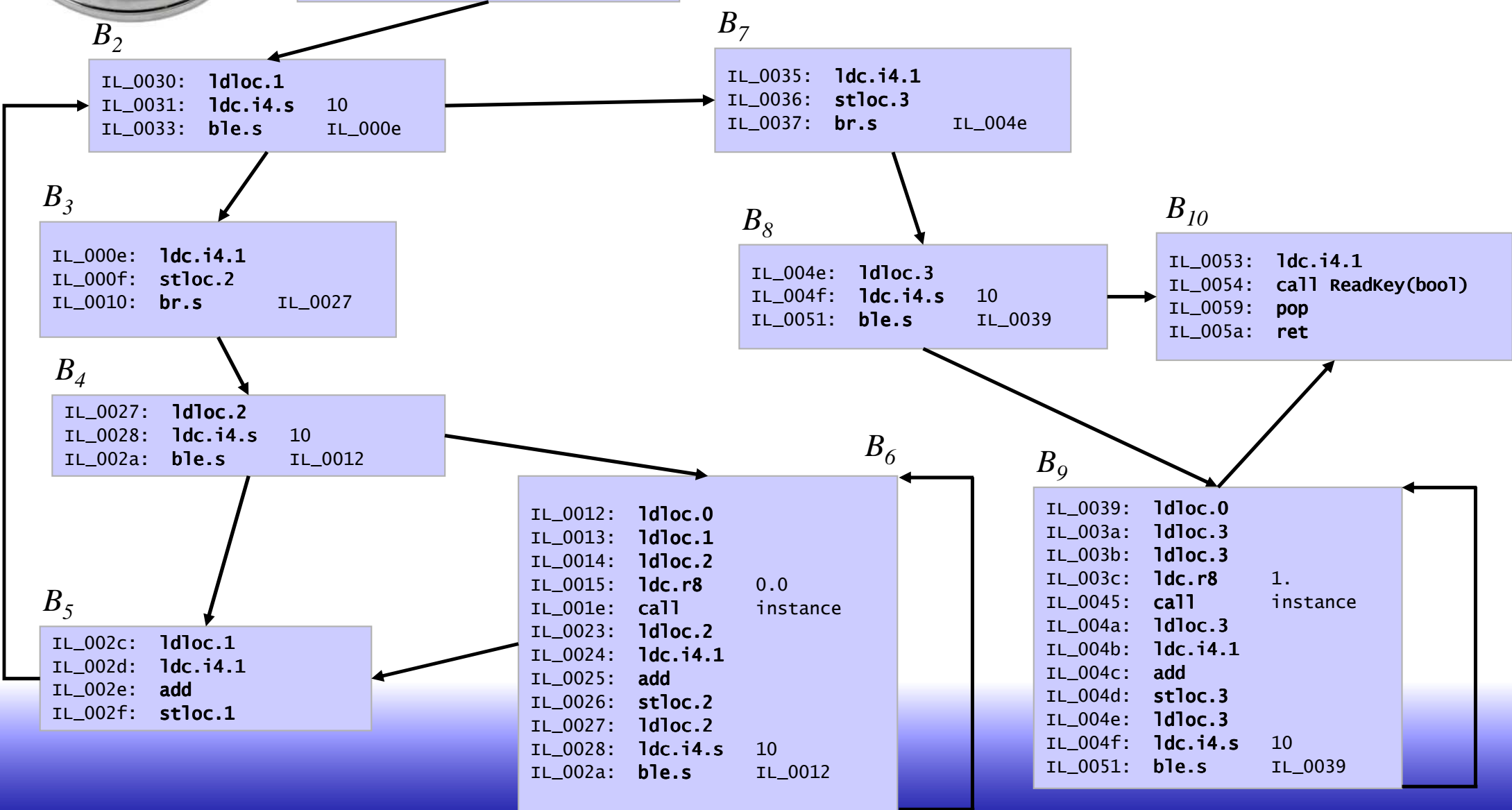
```
IL_0012: ldloc.0
IL_0013: ldloc.1
IL_0014: ldloc.2
IL_0015: ldc.r8 0.0
IL_001e: call instance
IL_0023: ldloc.2
IL_0024: ldc.i4.1
IL_0025: add
IL_0026: stloc.2
IL_0027: ldloc.2
IL_0028: ldc.i4.s 10
IL_002a: ble.s IL_0012
```

B_9

```
IL_0039: ldloc.0
IL_003a: ldloc.3
IL_003b: ldloc.3
IL_003c: ldc.r8 1.
IL_0045: call instance
IL_004a: ldloc.3
IL_004b: ldc.i4.1
IL_004c: add
IL_004d: stloc.3
IL_004e: ldloc.3
IL_004f: ldc.i4.s 10
IL_0051: ble.s IL_0039
```

B_5

```
IL_002c: ldloc.1
IL_002d: ldc.i4.1
IL_002e: add
IL_002f: stloc.1
```



Applications of CFG



For implementation of optimization methods

- ❖ Dead Code Elimination
- ❖ Removing of unconditional branches
- ❖ ...

For analysis

- ❖ Code coverage
- ❖ Hotspots

Implementation



CLR has specific instructions for exception handling

→ we need nested blocks

Design pattern “Composition”

- ❖ Allows composite elements in the graph (exceptions)



Analogous to CFG, which

- ❖ There is one node for each subroutine in the program
- ❖ There is one node for each call site
- ❖ If call site may call subroutine, there is an edge between those two nodes

Applications of CGs



Applications

- ❖ Simplifies method cloning/inlining
- ❖ Simplifies the method devirtualization
- ❖ Simplifies the parallelization
- ❖ Interprocedure analysis
- ❖ ...

Three-address code



Sequence of instructions of type:

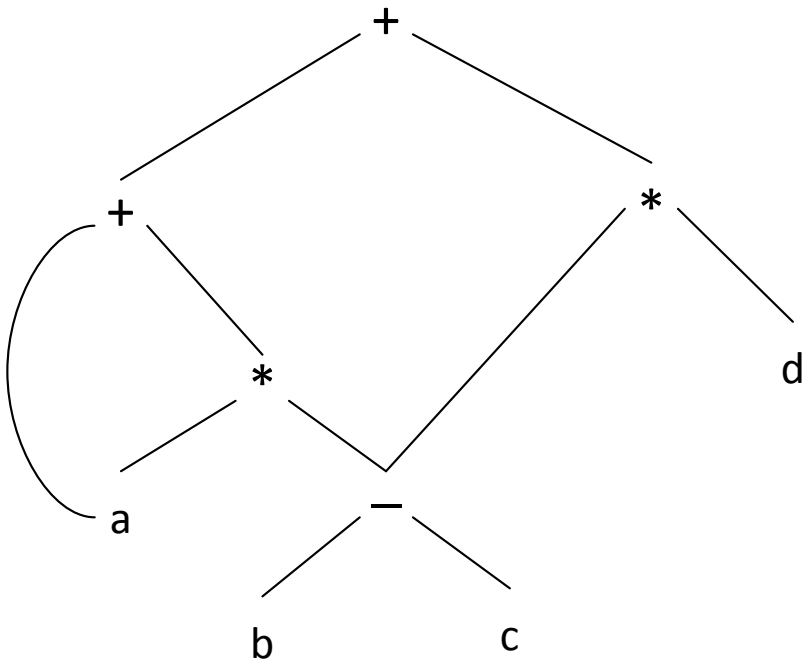
- ❖ For Expressions
 - ❖ $x = y \text{ op } z$
 - ❖ $x [y] = z$
 - ❖ $x = y [z]$
- ❖ For Statements
 - ❖ `ifFalse x goto L`
 - ❖ `ifTrue x goto L`
 - ❖ `goto L`
- ❖ For copying
 - ❖ $x = y$

x, y, z – addresses;
 op – binary operator | logical operator
 $=$ – copy

Three-address code



$a + a * (b - c) + (b - c) * d$



Directed Acyclic Graph

```
t1 = b - c
t2 = a * t1
t3 = a + t2
t4 = t1 * d
t5 = t3 + t4
```

Three-Address Code

How to implement the three-address code. Labels



```
do i = i + 1; while (a[i] < v);
```

```
L:  t1 = i + 1  
    i = t1  
    t2 = i * 8  
    t3 = a [ t2 ]  
    if t3 < v goto L
```

Symbolic labels

```
100:  t1 = i + 1  
101:  i = t1  
102:  t2 = i * 8  
103:  t3 = a [ t2 ]  
104:  if t3 < v goto 100
```

Position numbers



How to implement the three-address code. Quadruples

```
a = b * -c + b * -c;
```

```
t1 = minus c  
t2 = b * t1  
t3 = minus c  
t4 = b * t3  
t5 = t2 + t4  
a = t5
```

Three-address code

| | <i>op</i> | <i>arg₁</i> | <i>arg₂</i> | <i>result</i> |
|---|-----------|------------------------|------------------------|----------------|
| 0 | minus | c | | t ₁ |
| 1 | * | b | t ₁ | t ₂ |
| 2 | minus | c | | t ₃ |
| 3 | * | b | t ₃ | t ₄ |
| 4 | + | t ₂ | t ₄ | t ₅ |
| 5 | = | t ₅ | | a |
| | | | ... | |

Quadruples

How to implement the three-address code. Triples



`a = b * -c + b * -c;`

| | <i>op</i> | <i>arg₁</i> | <i>arg₂</i> |
|---|-----------|------------------------|------------------------|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | | ... | |

Triples

| | <i>instruction</i> |
|----|--------------------|
| 35 | (0) |
| 36 | (1) |
| 37 | (2) |
| 38 | (3) |
| 39 | (4) |
| 40 | (5) |
| | ... |

| | <i>op</i> | <i>arg₁</i> | <i>arg₂</i> |
|---|-----------|------------------------|------------------------|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |
| | | ... | |

Indirect triples



Single Static Assignment

$e - (a + b - c) * d + a + b - c$

```
p = a + b
q = p - c
p = q * d
p = e - p
q = p + q
```

Three-address code

```
p1 = a + b
q1 = p1 - c
p2 = q1 * d
p3 = e - p2
q2 = p3 + q1
```

SSA Form

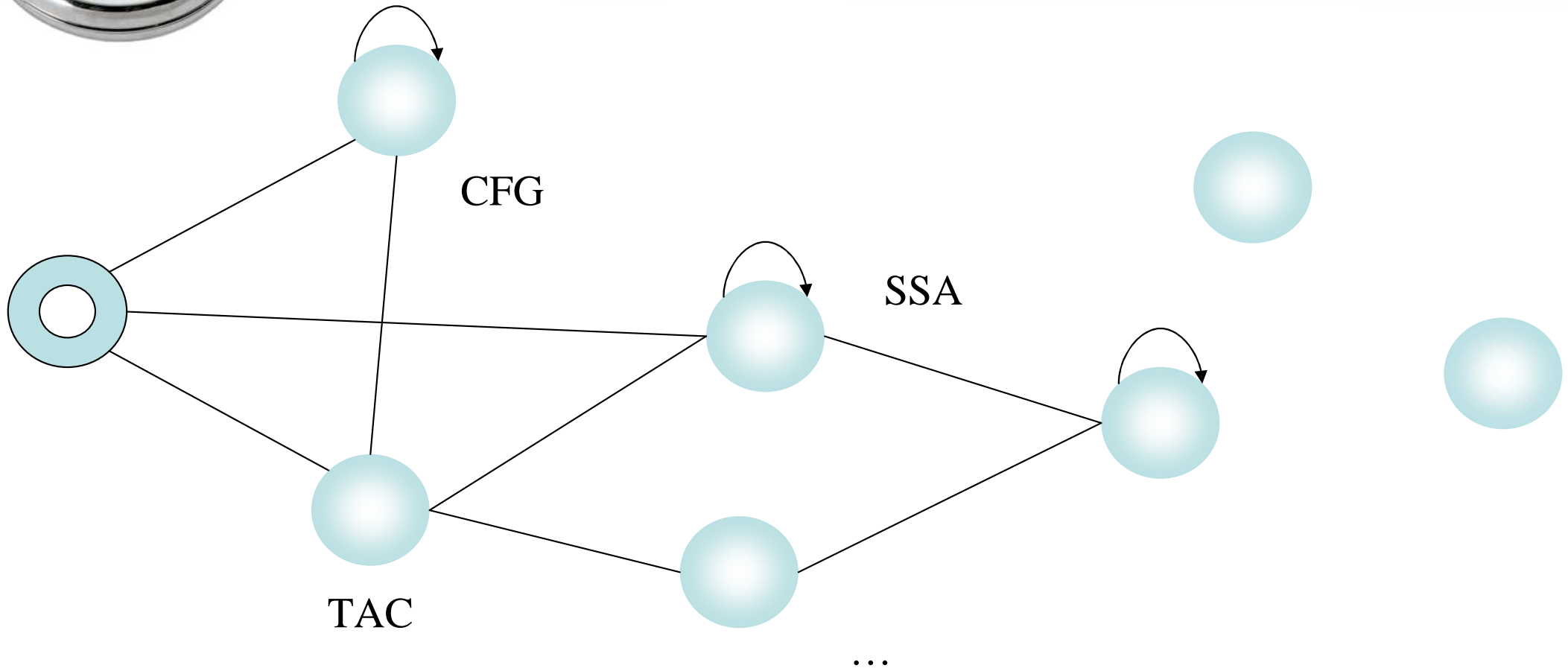
```
if (flag) x = -1; else x = 1;
y = x * a;
```



```
if (flag) x1 = -1; else x2 = 1;
y = φ(x1 x2) * a;
```

Conclusion

Applications of the notations



SolidOpt Notation Graph

Questions ?

Discussion...



Vasil.Georgiev.Vasilev@cern.ch



apenev@uni-plovdiv.bg