*) Named after a small passerine bird known for walking head-first towards the root of a tree on the underside of branches.

http://nuthatchery.org/

https://github.com/nuthatchery/nuthatch

Nuthatch*/J – A tree walk DSL/Library

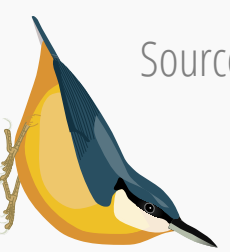
Based on: „Walk your tree any way you want“, A. H. Bagge and R. Lämmel, June 2013

SLE Winter Term 2015/16, Ass. 02, University of Koblenz-Landau

Maximilian Strauch (Dec 10th, 2015)

UNIVERSITÄT KOBLENZ · LANDAU

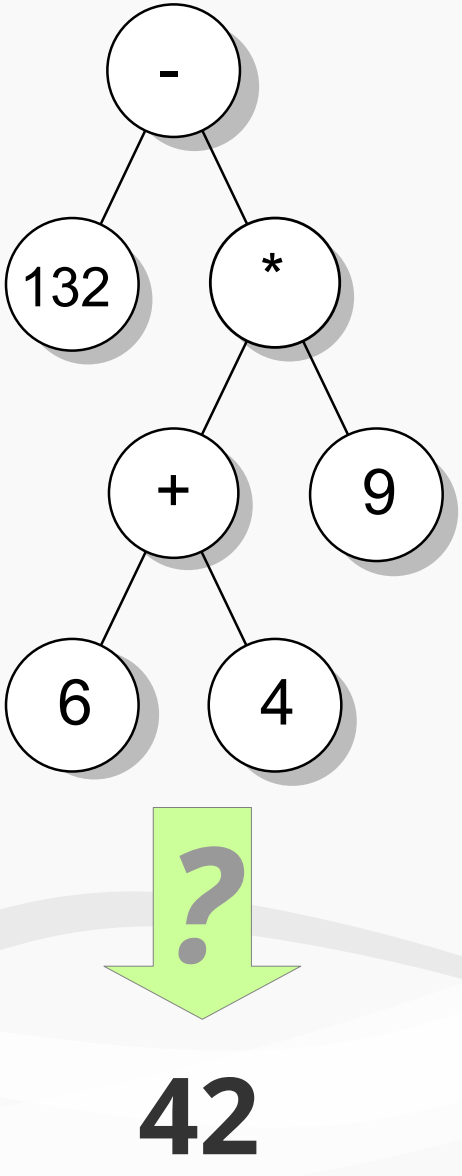
http://softlang.wikidot.com/course:sle1516



Source: [1], [2]

Motivation: why a tree walk library?

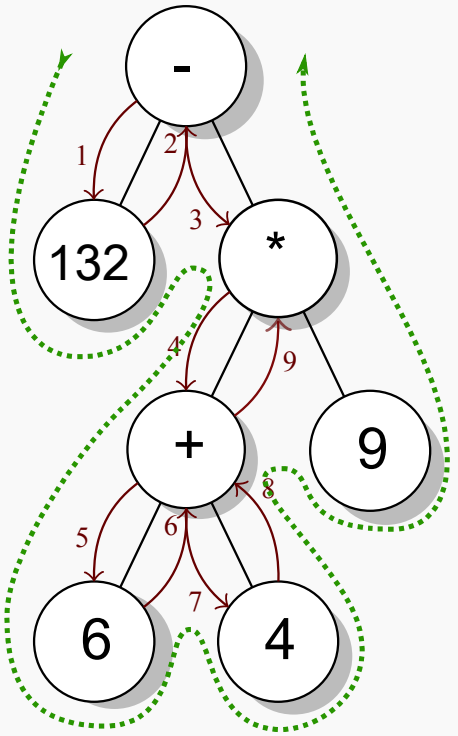
- ***Objective:*** *go through the tree and calculate 42 (aka „eval“ the tree)!*
- Tree Walks:
  - Step-wise traversal through the tree
  - State-access during traversal
  - A “walk“ abstracts over a traversal through the tree Implemented using Java

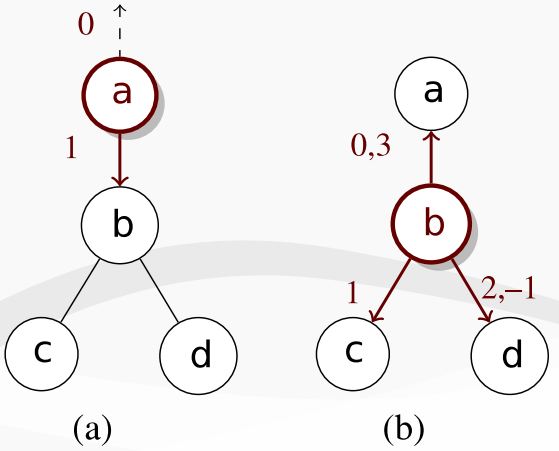


Source: [1]

What's a walk?

- A **walk** *walks* along a tree, selects branches and mutates nodes (rewriting)
- Path: sequence of nodes; default Path: f–2–f–*–+–6– . . .
- If a walk comes to a node step() is performed
  - Join point captures enter condition
  - Return value = next node

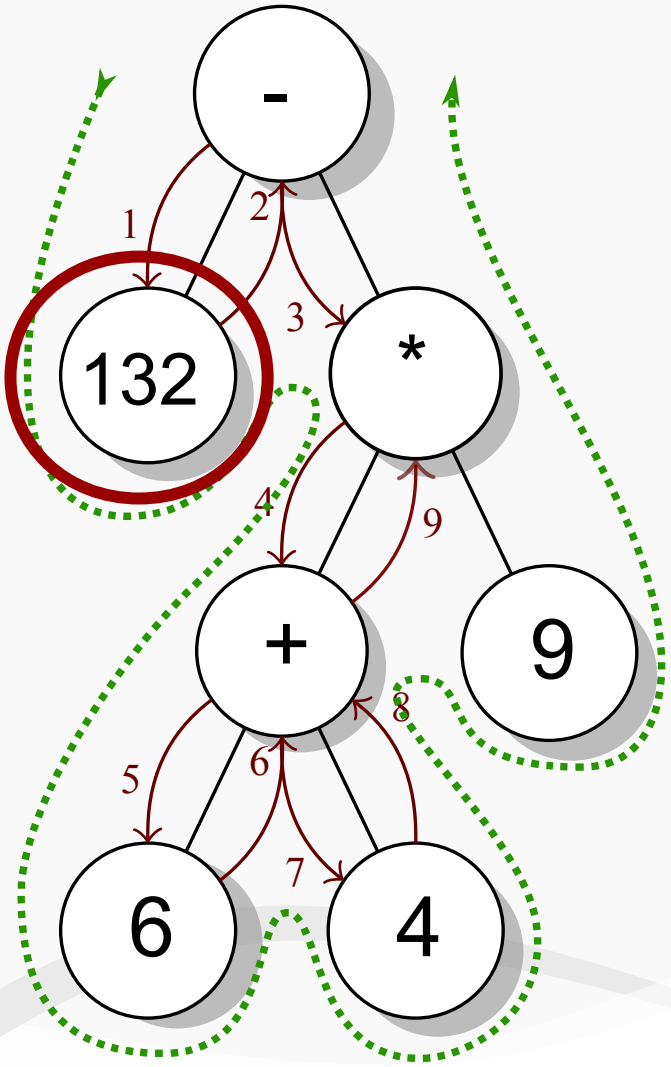




Source: [1]

Join Points

- If a node is visited along the *path*, certain conditions apply for this node
- Example Join Points:
  - `down ⇔ from == 0`
  - `up ⇔ isLeaf || from == arity`
- For node “132“:
  - **down** (from == 0)
  - **up** (isLeaf == true, arity == 0)



Source: [1]

A DSL for tree walks

- [1] proposes a custom DSL to define tree walks *(only one simple example given here!)*

Stateful variable definition

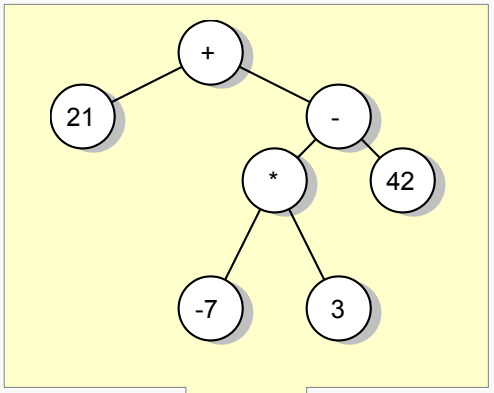
```
walk toString {
  state s = "";
  if leaf {
    then s += data;
  } else {
    if down then s += name + "(";
    if up then s += ")";
    if from>=first && from<last then s += ", ";
  }
  walk to next;
}
```

Various Join Points reacting to different node "events"

Memory of current and last visited node

Define where to go next; here: default walk.
Possible other expression:

```
walk to (if leaf then parent else last);
```




```
+(21, -(*(-7, 3), 42))
```

https://github.com/nuthatchery/nuthatch

Implementation in Java (Nuthatch/J)

```
final StringBuffer s = new StringBuffer();
```

Statefull data storage

```
Walk<SimpleWalker<String, String>> toTerm = new BaseWalk<SimpleWalker<String, String>>() {
    public int step(SimpleWalker<String, String> w) {
        if (leaf(w))
            s.append(w.getData());
        else
            if (down(w))
                s.append(w.getName() + "(");
            else if(up(w))
                s.append(")");
            else
                s.append(", ");
        return NEXT;
    }
};
```

Step method is executed, if a node is visited; Join Points can be used to determine further information about the node

```
new SimpleWalker<String, String>(TREE.makeCursor(), toTerm).start();
```

Create a walker (and run it) with a sample tree and the step "listener"

```
System.out.println(s.toString());
```

Result output

Source: [1]

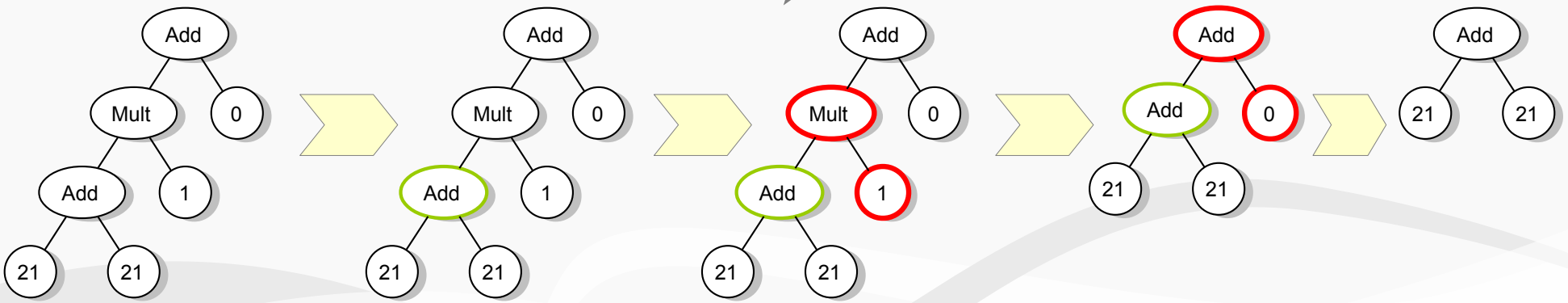
Tree mutation

- Rewrite rules are encoded with
  - Match condition (“?“)
  - Replacement Action (“!“)

Rules:
- `Add(x, 0)` → x (unit law)
- `Mult(x, 1)` → x (zero law)
- …

```
walk simplify {
  if up then {
    if ?Add(x, 0) then !x;
    if ?Mult(x, 0) then !0;
  }
  walk to next;
}
```

```
Add(Mult(Add(21,21),1),0)
```



Source: [2]

Pattern matching

- Nodes can be matched by name, type, data, children, parent, ancestors or *any combination*
- A pattern might have a variable which then gets bound and can be accessed e.g. to rewrite

```
public int step(ExprWalker w) {
   if (down(w)) {
      if (w.match(Add(var("x"), Int(0))))
         w.replace(w.getEnv().get("x"));
      if (w.match(Mul(var("x"), Int(0))))
         w.replace(Int(0));
   }
   return NEXT;
}
```

Access the now bound variable x

Rules from example Before applied as pattern

Thank you for your attention.

Any Questions?

SLE Winter Term 2015/16, University of Koblenz-Landau

Maximilian Strauch

UNIVERSITÄT KOBLENZ · LANDAU

http://softlang.wikidot.com/course:sle1516

BACKUP

References

- [1] A. H. Bagge and R. Lämmel. Walk your tree any way you want. 6th Int'l Conf on Model Transformation (ICMT'13), June 2013.
- [2] A. H. Bagge. Analysis and Transformation with the Nuthatch Tree-Walking Library. SLE Conference 2015, 2015.
- Further materials: http://nuthatchery.org/
- *Sitta Cashmirensis* imagery: https://commons.wikimedia.org/wiki/File:SittaCashmirensis.svg

Source: [1], [2]

Trees, graphs & cursors

- Mainly talked about **trees**, but **graphs** can be processed the same way
- **Tree handle** *approach* to abstraction:
  - A **tree builder** generates the tree and provides an interface to it
  - A **cursor** points to the current node in the tree and can be advanced (e.g. `go(int i)`)
- A **tree handle** abstracts over the used data structure and allows to interface with different systems (Stratego/XT, TXL, Tom, Rascal, …)

Source: [1]

http://nuthatchery.org/icmt13/benchmarks.html

Performance

- Average of 5k runs
- Comparison also to hand-written Java code (optimal performance)
- Not yet optimized for performance!
- Rewriting is very fast with Nuthatch/J
- Compiled Stratego is very fast

Nuthatch/J vs. others