Understanding Git with Alloy

Milestone 2

Cláudio Lourenço Renato Neves

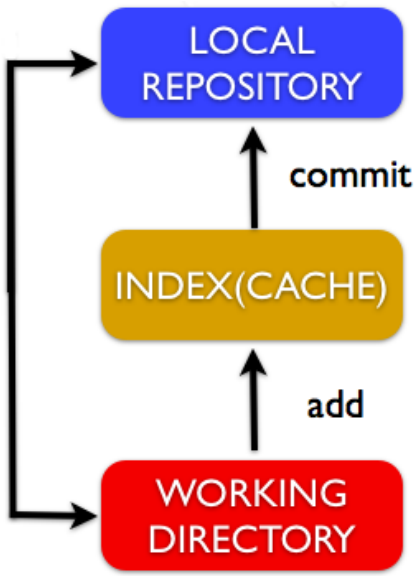
University of Minho
Formal Methods in Software Engineering

June 12, 2012

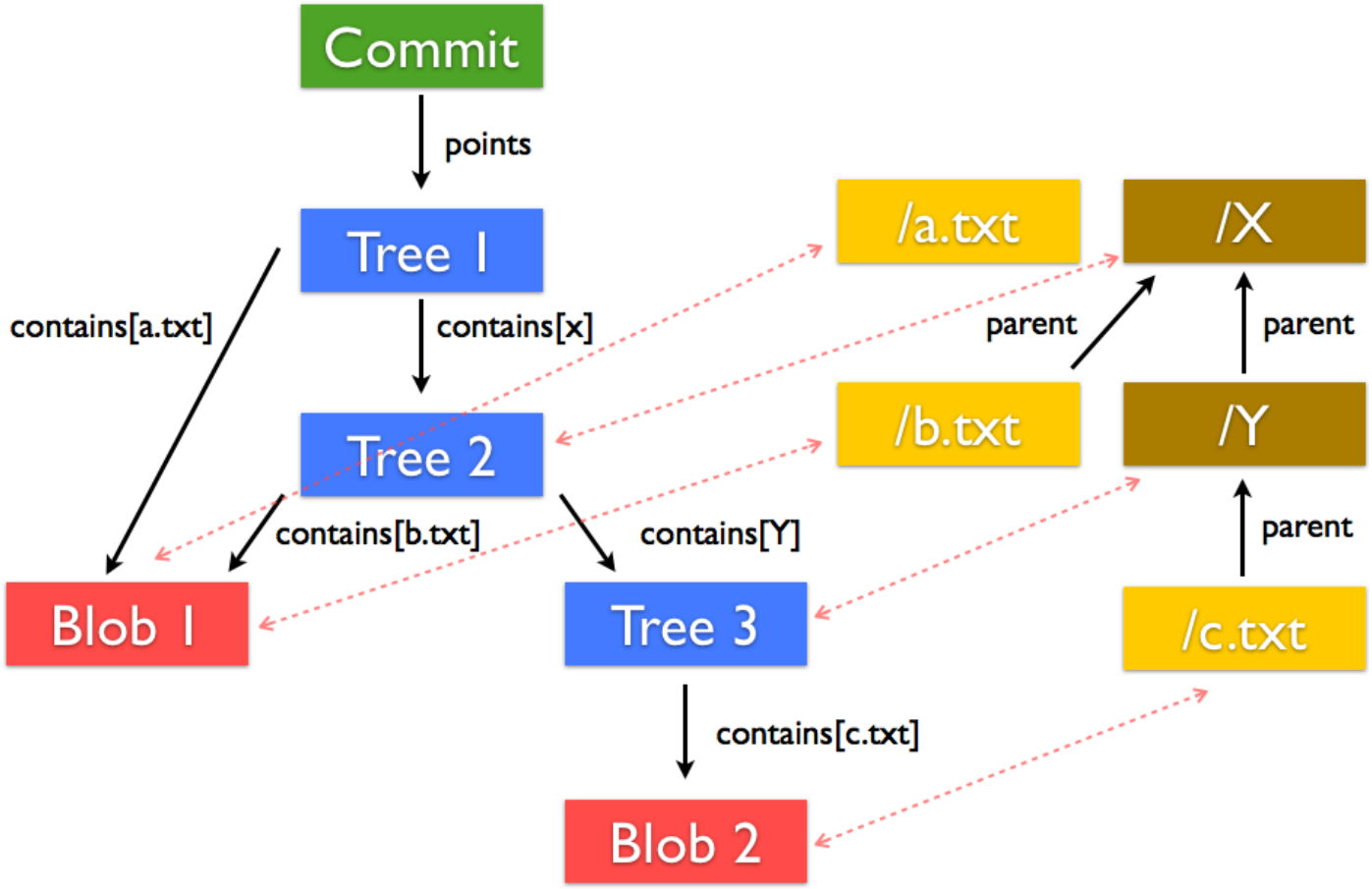
Table of contents

Where were we?

- Focusing on Index + Object Model
- No problem with add and rm operations
- Commit - Big Problem



The problem


Commit
points
Tree 1
contains[a.txt]
contains[x]
Tree 2
contains[b.txt]
contains[Y]
Blob 1
Tree 3
contains[c.txt]
Blob 2
/a.txt
/X
parent
parent
/b.txt
/Y
parent
/c.txt

Model

```
abstract sig Object {
   objects: set State
}

sig Blob extends Object {}

sig Tree extends Object {
   contains : Name -> lone (Tree+Blob)
}

sig Commit extends Object {
   points : Tree,
   parent : set Commit,
   abs: Path -> Object
}

sig RootCommit extends Commit {}

sig Branch{
   marks: Commit one -> State,
   branches: set State,
   head: set State
}

lone sig Master extends Branch{}
```

```
sig Path {
   pathparent : lone Path,
   name : Name
}

sig File{
   path: Path,
   blob: Blob,
   index: set State
}
```

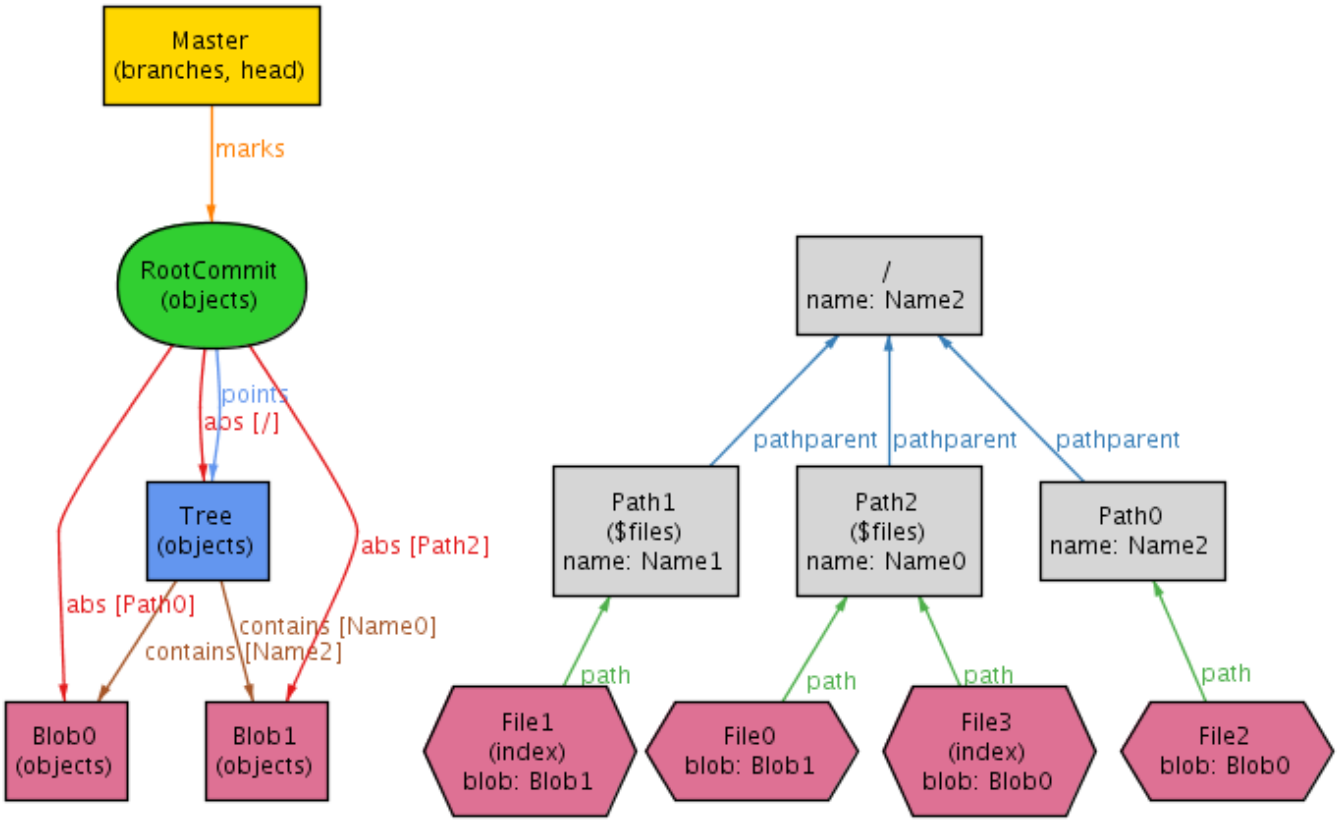
How did we solve it?

Abstraction

- In each commit there is a relation between Objects and Paths
- Facts to reflect the parent relationship
  - The Tree pointed by the Commit corresponds to the root
  - A contains tuple exists, if and only if, the corresponding parent tuple exists

```
sig Commit extends Object {
    points : Tree,
    parent : set Commit,
    abs: Path -> Object
}
```

The abstraction relation



CSAIL

Operations - add and rm

add

Add a file with the current content to the index

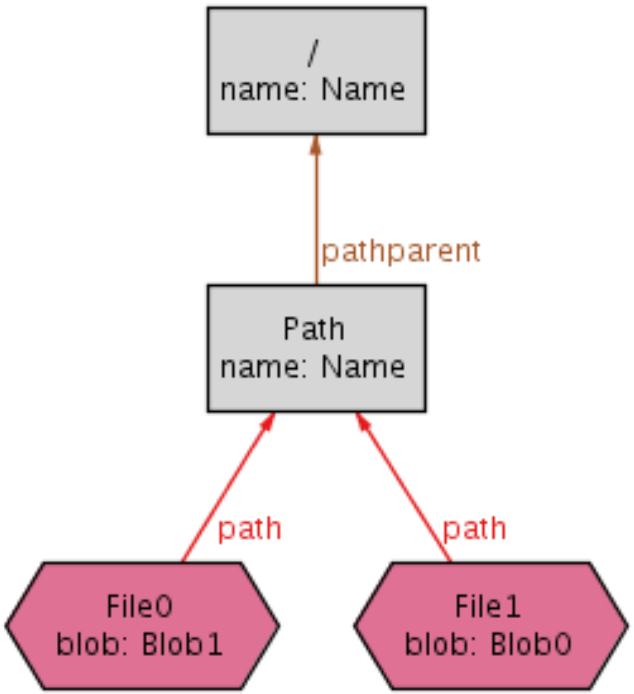
```
...
index.s' = index.s +
           f - ((f.path).~path -f)
...
```

rm

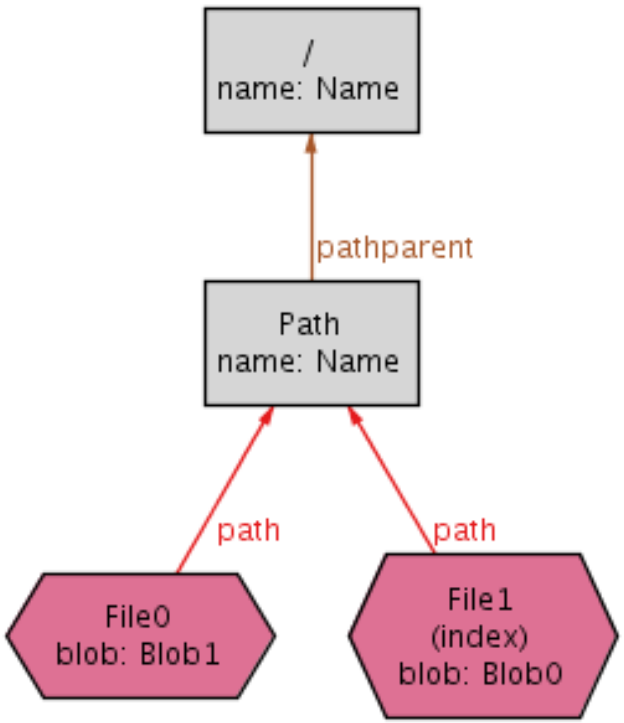
Remove the file from the index

```
...
index.s' = index.s - f
...
```

Operations - add and rm



Operations - add and rm



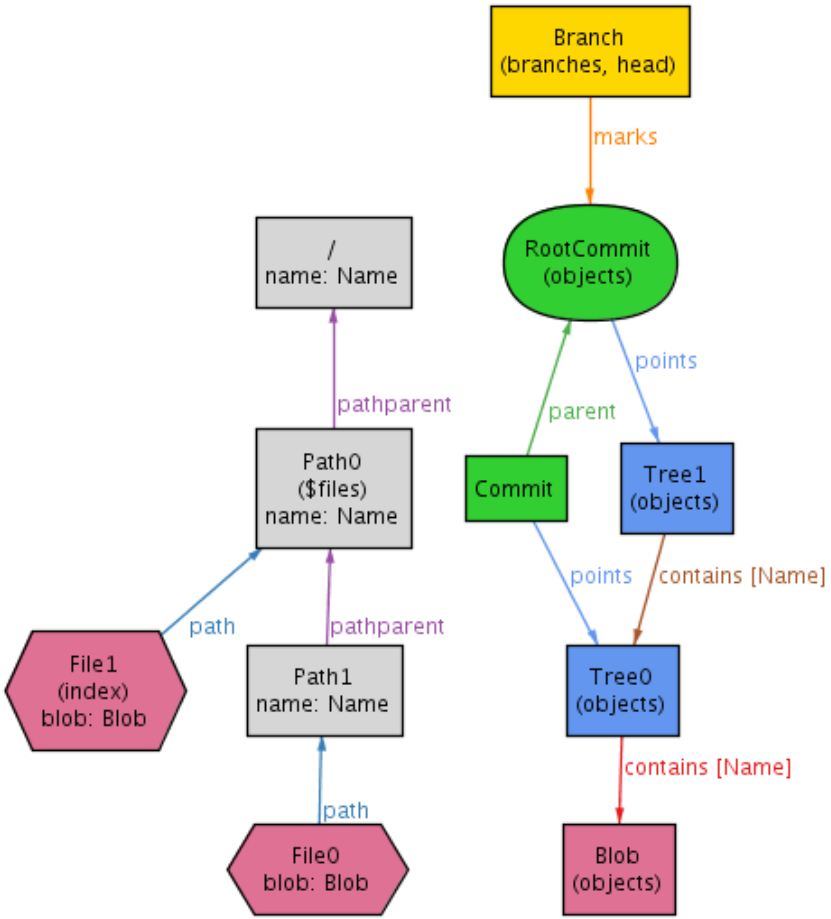
Operations - commit

commit

Creates a commit, from the index

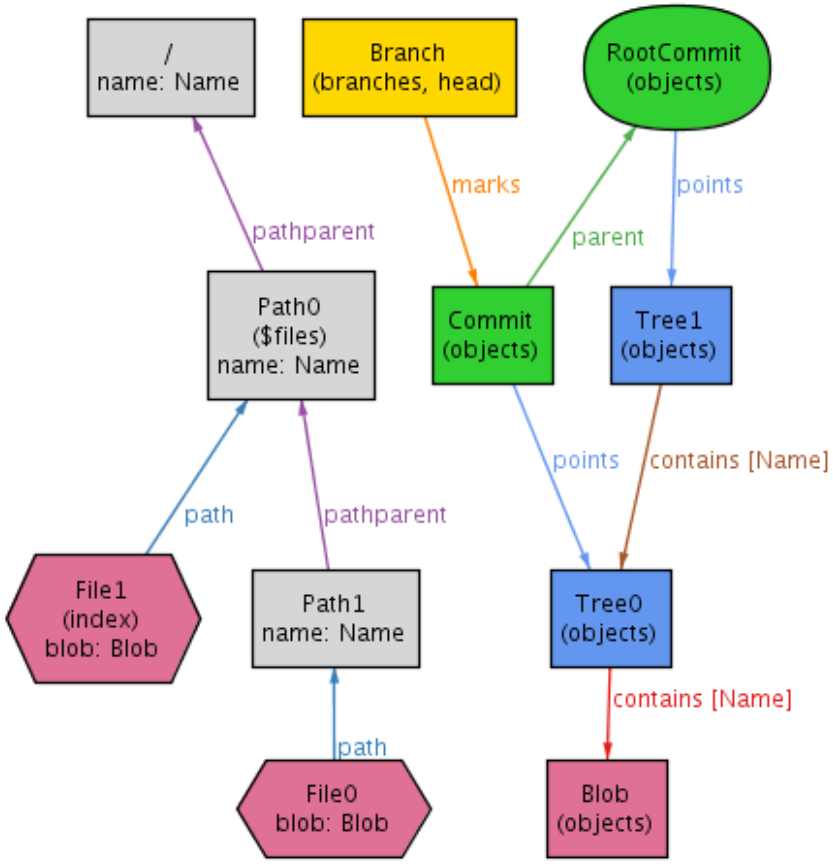
```
...
(head.s').(marks.s').parent = (head.s).(marks.s)
...
(index.s).path.*pathparent = (head.s').(marks.s').abs.univ
all f:index.s | f.path -> f.blob in (head.s').(marks.s').abs
...
```

Operations - commit



CSAIL

Operations - commit



Operations - branch

branch

Creates a new branch pointing to the current commit

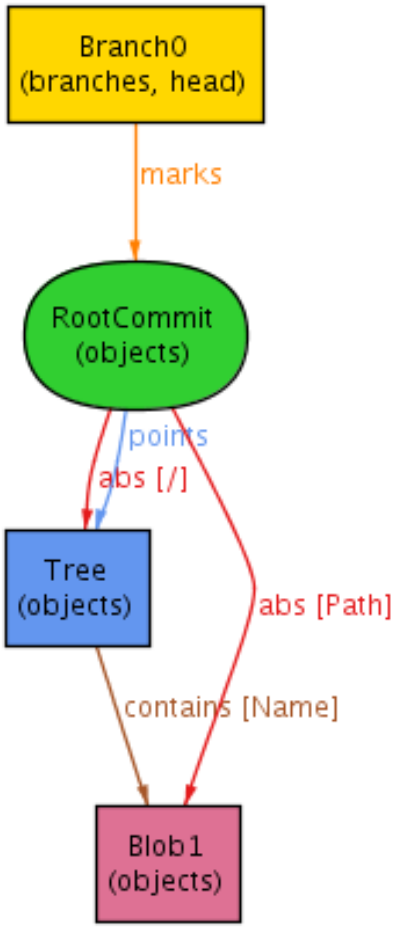
```
...
branches.s' = branches.s + b
marks.s' = marks.s + b -> (head.s).(marks.s)
...
```

branch -d

Removes a branch if it is not pointed by the head. Also it's information must be achieved by the current branch

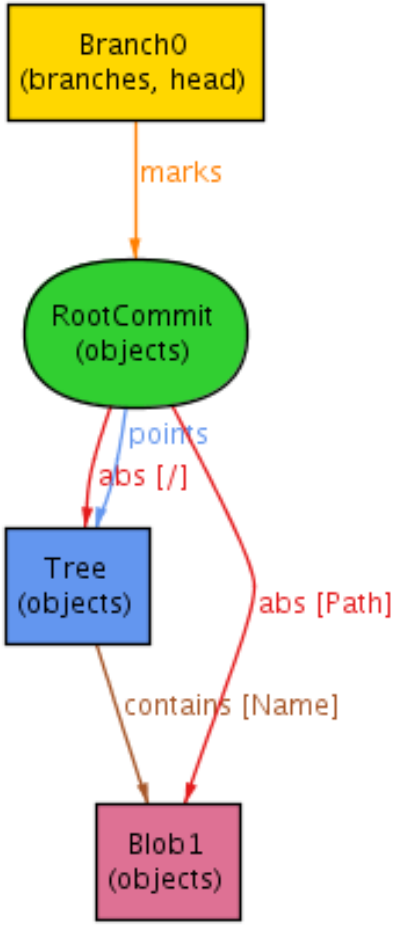
```
...
b not in (head.s)
b.marks.s in (head.s).(marks.s).*parent
...
branches.s' = branches.s - b
marks.s' = marks.s - b -> Commit
...
```

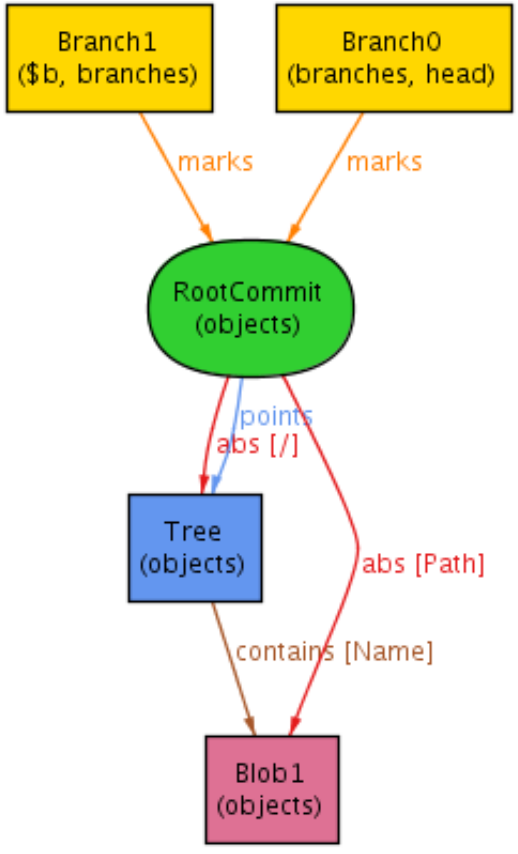
Operations - branch



CSAIL

Operations - branch





Operations - checkout

Updates the head and the index to reflect a certain commit pointed by a branch

Problems

- Difficulty to understand the pre-conditions
- There are no specifications
- "if there are any uncommitted changes when you run git checkout, Git will behave very strangely."
- "...Git resets your working directory to look like the snapshot of the commit that the branch you check out points to. It adds, removes, and modifies files automatically to make sure your working copy is what the branch looked like on your last commit to it."

Pre-conditions found relatively to the index

- Removed files are ignored
- Non modified files since the last commit are ignored
- New files or Modified files:
  - if file exists on the destination commit then:
    - the content is the same as in the index, or
    - the content in the destination commit is the same as in the current commit
  - if the file does not exist in the destination commit
    - it also does not exist on the current commit

```
all f : index.s | f.path -> f.blob in ((CA.univ <: IA) - CA)
and f.path in CB.univ =>
(f.path -> f.blob in CB or (f.path).CA = (f.path).CB)
```

- For all newly modified files, if their paths are also on CB then, or the new changes are already on CB or the old content it's equal on both commits

Operations - checkout

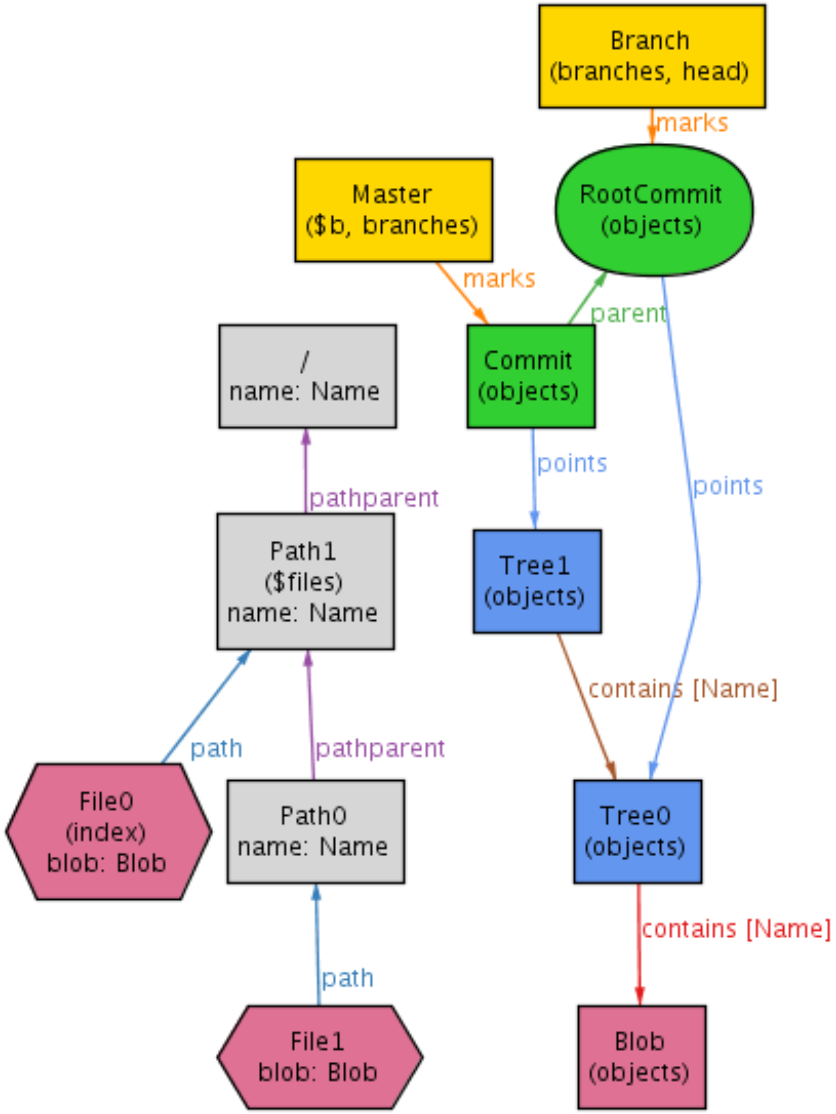
Master

- 1 - g (add), commit
- 2 - g (rm)
- 3 - g/f (add)
- 4 - git checkout B
- 6 - All that was inside the folder g disappeared

B

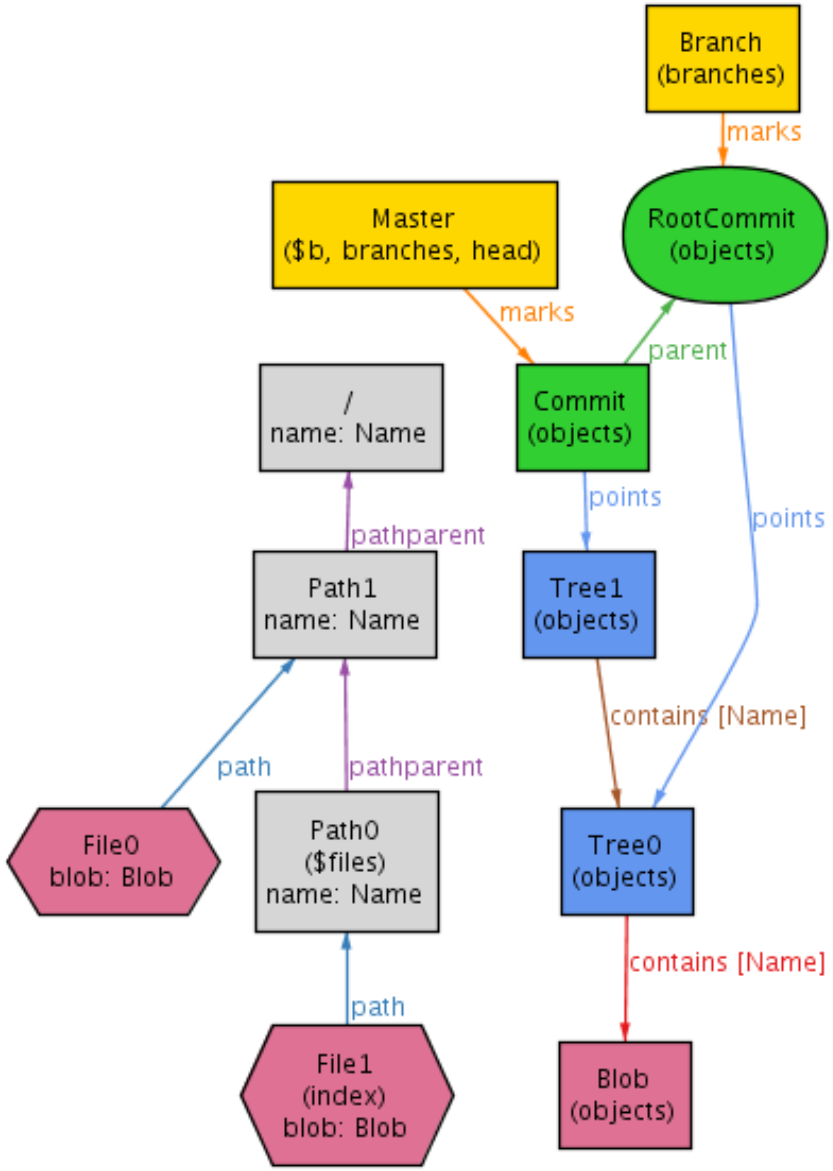
- 5 - git checkout Master

Operations - checkout



CSAIL

Operations - checkout



CSAIL

Properties

Invariant preservation

All operations must preserve the invariant

all s,s':State,... | invariant[s] and operation[s,s',...] => invariant[s']

```
pred invariant[s:State]{
    some Commit & objects.s <=> some marks.s && one head.s
    head.s in branches.s & (marks.s).Commit
    (objects.s - Commit) in (objects.s).points.*(contents.Name)
    all t : objects.s & Tree | t.contents.Name in objects.s
    all c:objects.s & Commit
      | c.points in objects.s and c.parent in objects.s
    marks.s in branches.s -> one objects.s
    all s:State, t : objects.s & Tree | some t.contains
}
```

Properties

Idempotence

- After performing an operation, repeating it does not change the state
- Add, commit and checkout are idempotent

```
all s0,s1,s2 : State |
operation[s0,s1,...] and operation[s1,s2,...] =>
dynamicRelations[s1] = dynamicRelations[s2]
```

Properties

Commit, Add, Commit, Rm, Commit

- Resulting from this sequence of operations, the last commit must be equal to the first commit

```
all s0,s1,s2,s3,s4,s5:State, f:File |
(commit[s0,s1] and add[s1,s2,f] and
f.path not in (index.s1).path and
commit[s2,s3] and rm[s3,s4,f] and commit[s4,s5]) =>
((head.s1).(marks.s1).points = (head.s5).(marks.s5).points)
```

Properties

Revert the Checkout

- If we checkout to a given branch, and then checkout to the branch where we were, the commit and index before the first checkout must be the equal to the commit and index after the second checkout. In other words, no changes in the state

```
all s,s',s'' : State , b : Branch |
(checkout[s,s',b] and checkout[s',s'',head.s]) =>
(head.s).(marks.s) = (head.s'').(marks.s'') and
s.pathcontents = s''.pathcontents
```

Maybe show some simple workflow ?

Future work

- Finish the Merge operation
- Add more interesting properties
- Maybe try to model git rebase
- Document operations and properties

Understanding Git with Alloy

Milestone 2

Cláudio Lourenço Renato Neves

University of Minho
Formal Methods in Software Engineering

June 12, 2012