Understanding Git with Alloy

Milestone 2

Cláudio Lourenço Renato Neves

University of Minho
Formal Methods in Software Engineering

June 13, 2012

Table of contents

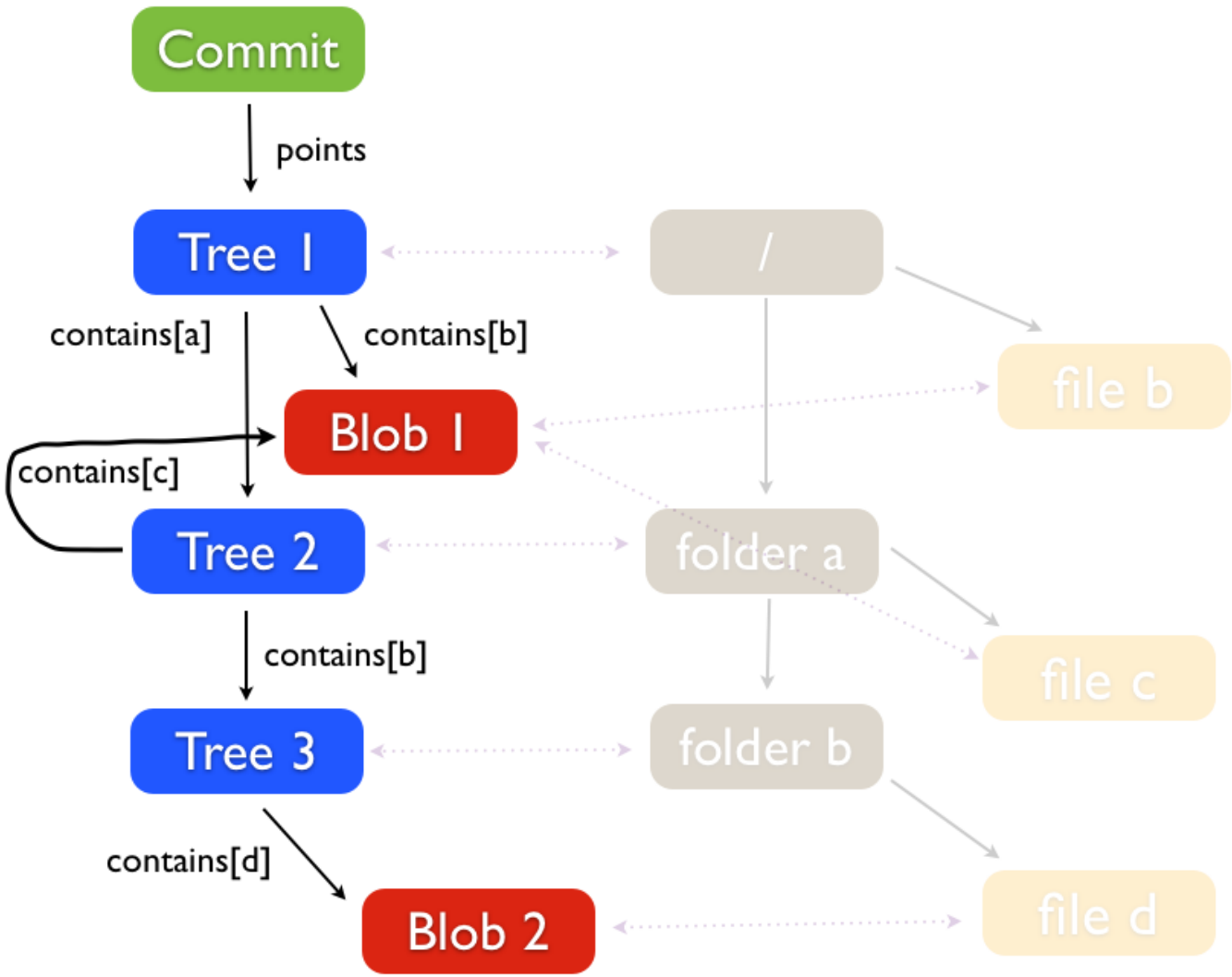
Where were we?

- Focus on Index and Object Model
- Ditched the Working Directory
- No problems with add and rm operations
- Big Problem in the commit

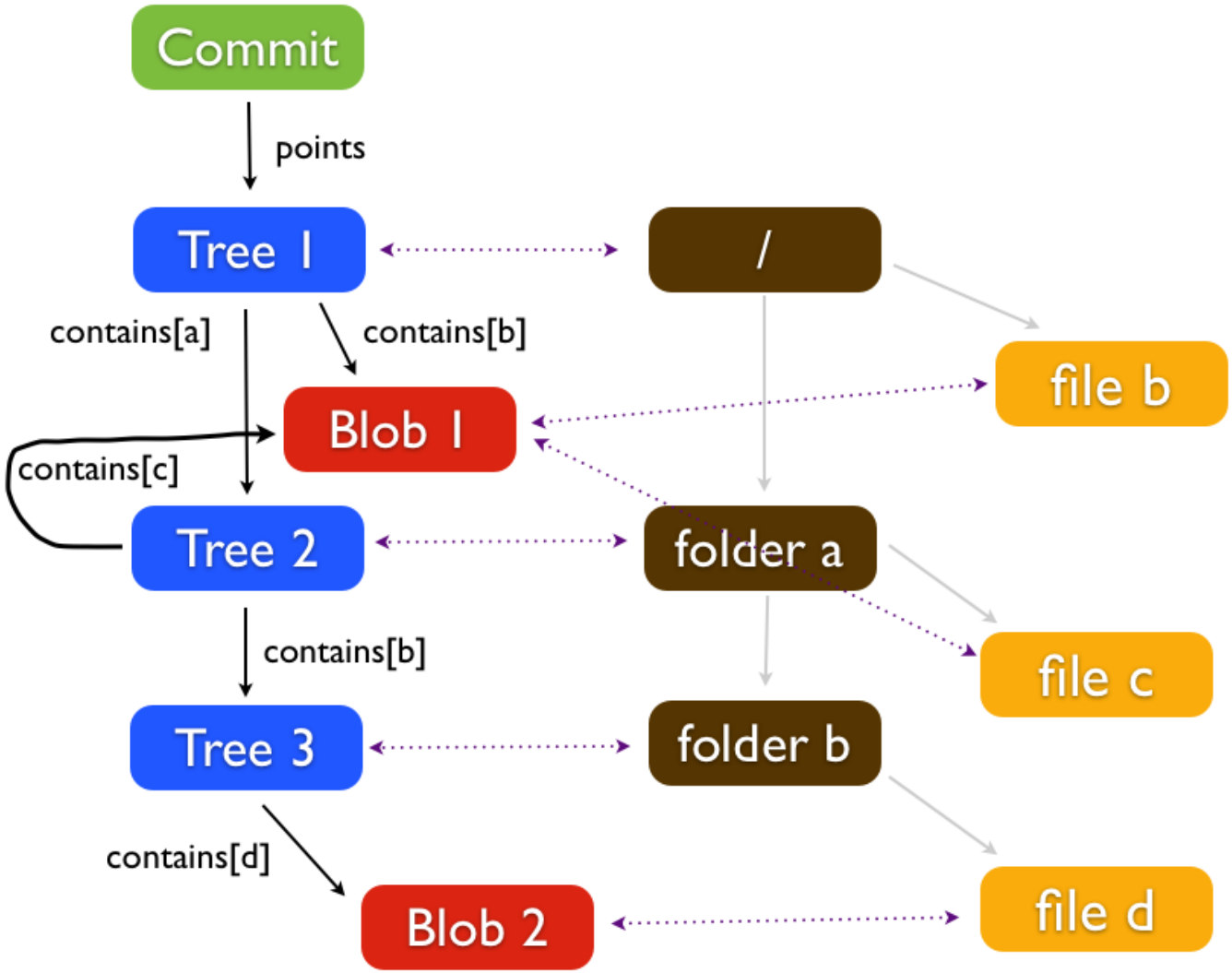
What we are going to show

- The solution to the commit problem
- Heads up of the current model
- Show Commit, Add, Rm and Branch operations
- Focus on the Checkout operation
- Show some properties
- Future work

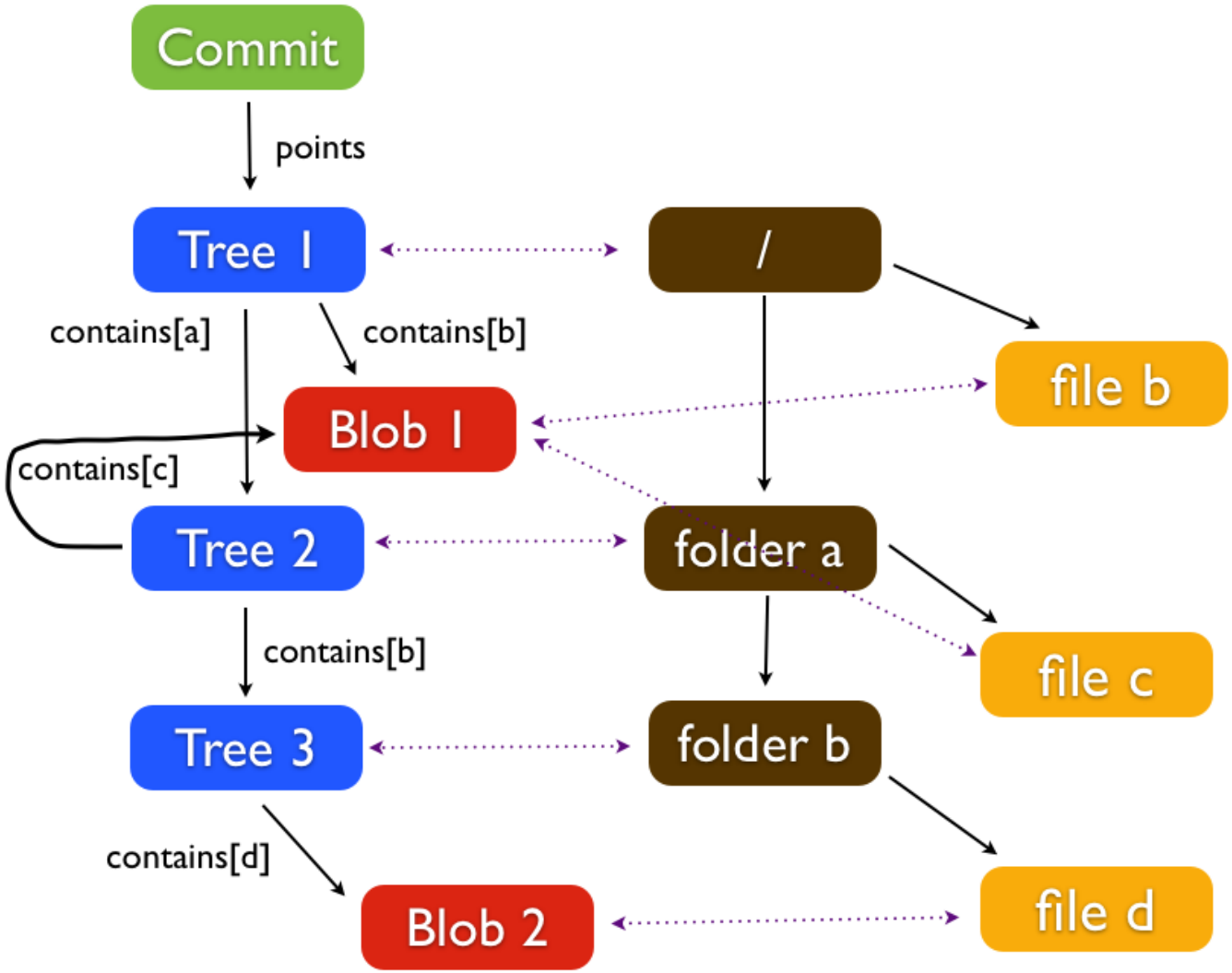
The problem


Commit
points
Tree 1
/
contains[a]
contains[b]
file b
Blob 1
contains[c]
Tree 2
folder a
file c
contains[b]
Tree 3
folder b
contains[d]
Blob 2
file d

The problem


Commit
points
Tree 1
/
contains[a]
contains[b]
file b
Blob 1
contains[c]
Tree 2
folder a
file c
contains[b]
Tree 3
folder b
contains[d]
Blob 2
file d

The problem


Commit
points
Tree 1
/
contains[a]
contains[b]
file b
Blob 1
contains[c]
Tree 2
folder a
file c
contains[b]
Tree 3
folder b
contains[d]
file d
Blob 2

Top view of the Model

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

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

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{}
```

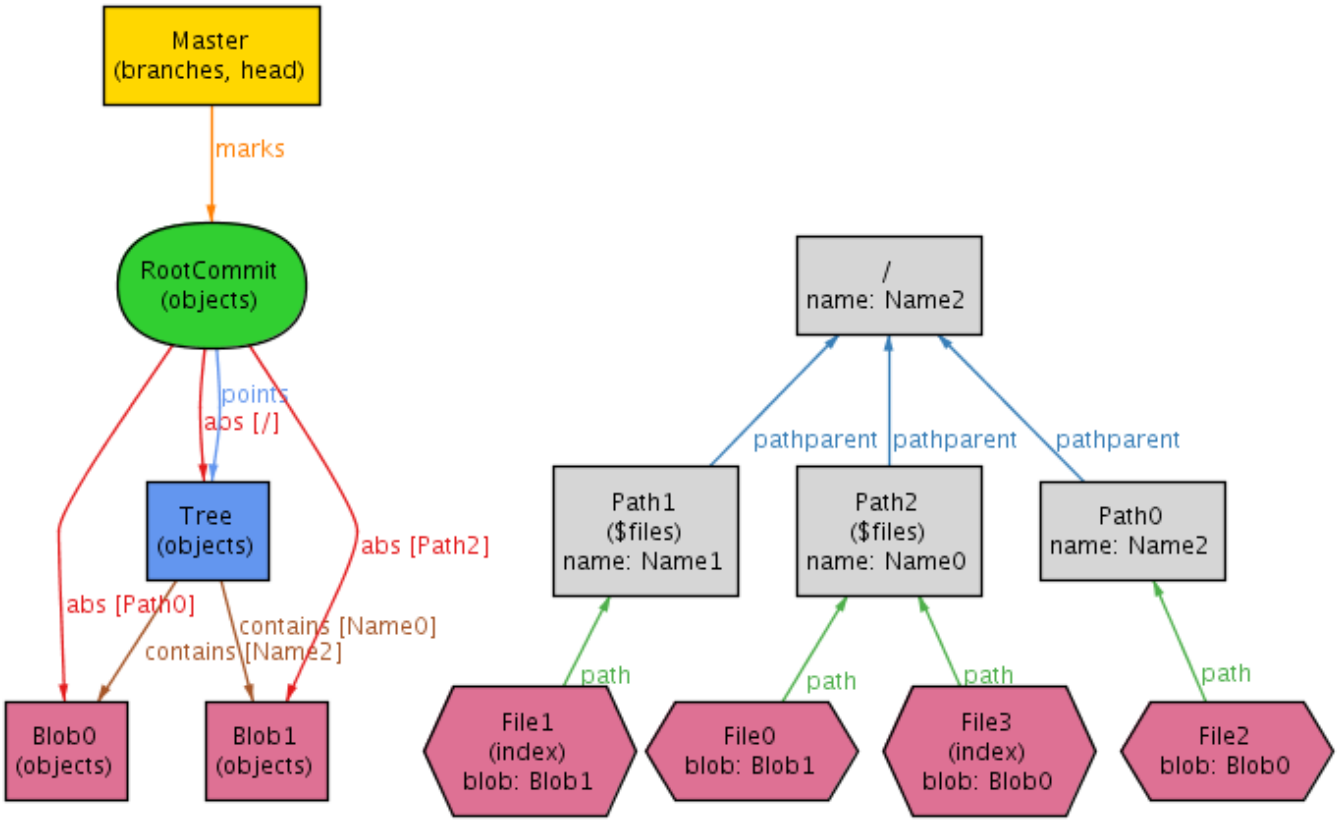
Commit

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 pathparent tuple exists

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

The abstraction relation



CSAIL

Operations - commit

Commit

Creates a commit object, using the index as source information

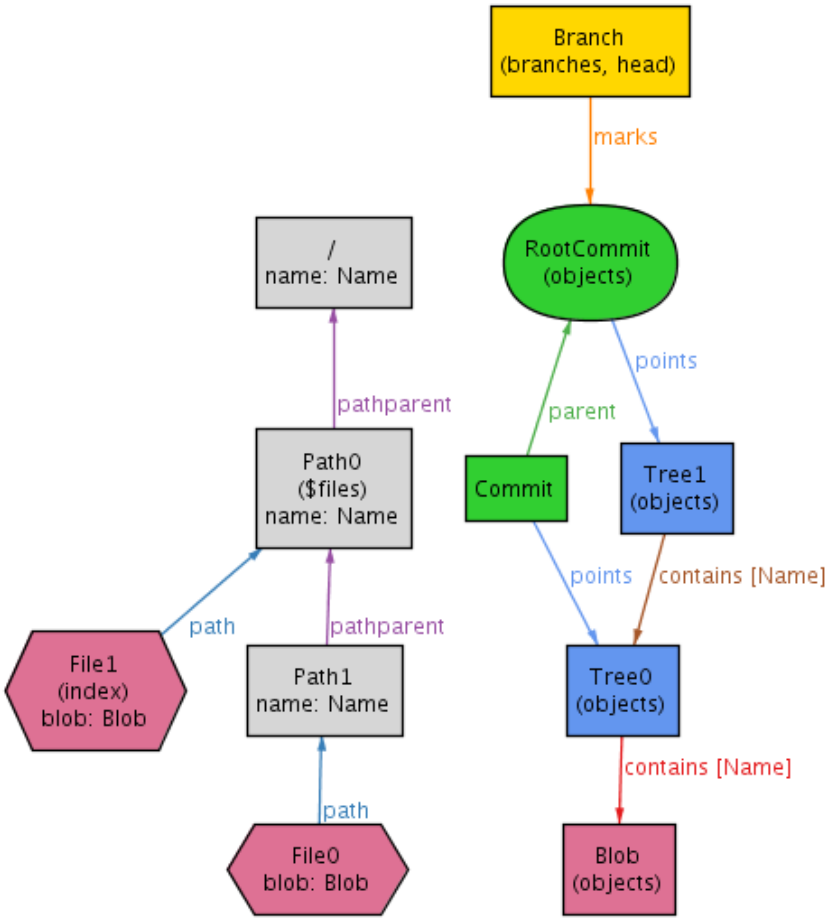
Commit Restrictions

```
all p,q : (c.abs).univ | p -> q in pathparent =>
q.(c.abs) -> p.(c.abs) -> p.name in contents
...
all t,o : objs, n : Name | t -> o -> n in contents =>
all y : c.abs.t | some x : c.abs.o | x -> y in pathparent and x.name = n
...
```

Commit Post Conditions

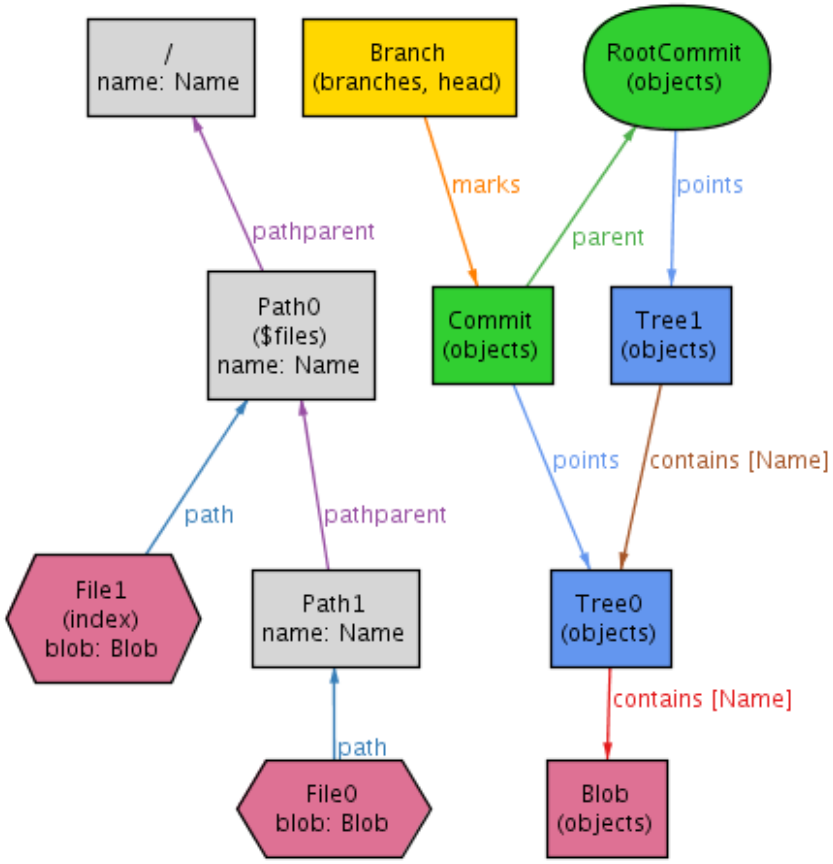
```
(head.s').(marks.s').parent = (head.s).(marks.s)
...
(index.s).path.*pathparent = (head.s').(marks.s').abs.univ
all f:index.s | f.path -> f.blob in (head.s').(marks.s').abs
...
```

Operations - commit



CSAIL

Operations - commit



Operations - 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

- The file must exist in the index
- The file with its content must exist in the current commit
- If you add a file, you can only remove it after committing it

```
...
f in index.s
f.path -> f.blob in (head.s).(marks.s).abs
...
index.s' = index.s - f
```

Operations - branch

branch

Creates a new branch pointing to the current commit

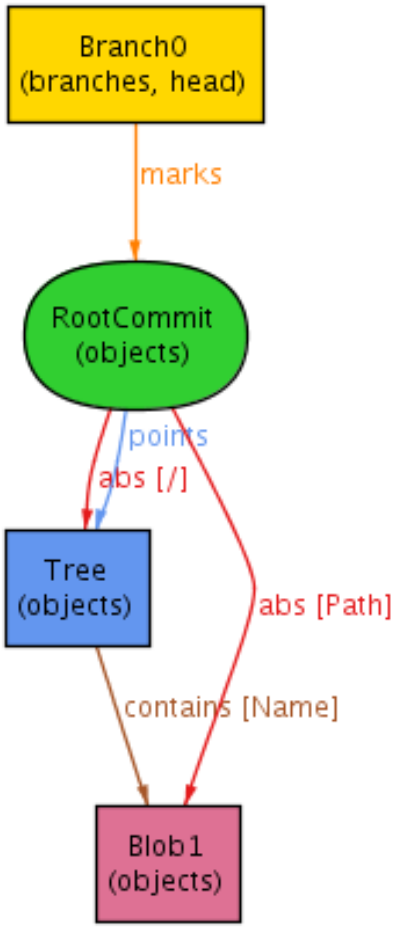
```
...
branches.s' = branches.s + b
marks.s' = marks.s + b -> (head.s).(marks.s)
...
```

branch -d

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

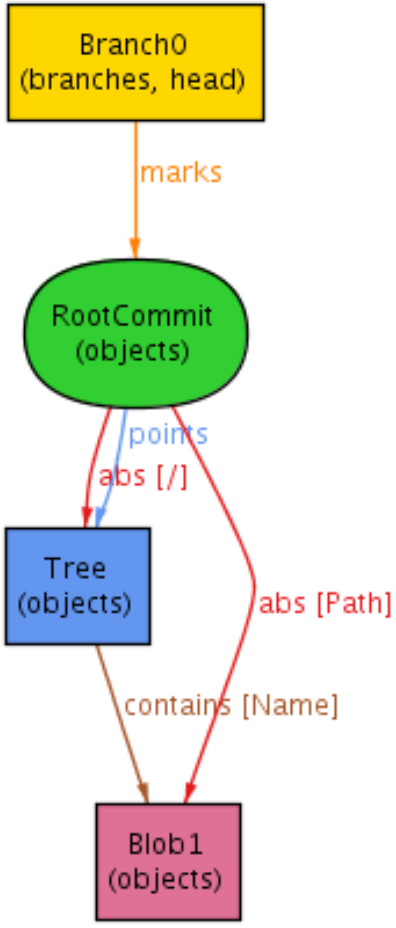
```
...
b not in (head.s)
b.marks.s in (head.s).(marks.s).*parent
...
branches.s' = branches.s - b
marks.s' = marks.s - b -> Commit
...
```

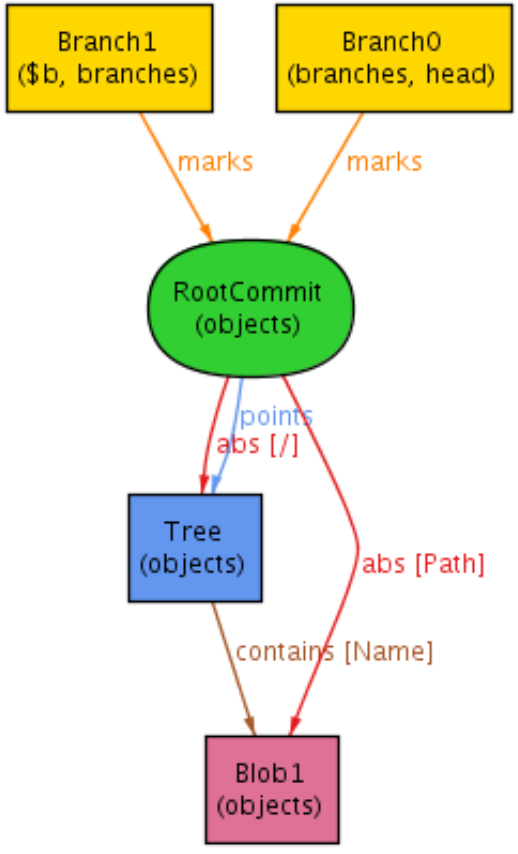
Operations - branch



CSAIL

Operations - branch





Operations - checkout

"...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." [1]

[1] Git Community Book
[2] Understanding Git

Operations - checkout

"...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." [1]

Problems

- There are no specifications
- Difficulty to understand the pre-conditions
- "if there are any uncommitted changes when you run git checkout, Git will behave very strangely." [2]

[1] Git Community Book
[2] Understanding Git

Checkout

Pre-conditions found

Everything that is in the index has to be in the current commit with the same content, except if:

- The content of a file is the same in the current and destination commit - in this case the file in the index keeps its content (warning is thrown)
- Exists a file in the index, and that file does not exists neither in the current nor in the destination commit - in this case the file is kept in the index (warning is thrown)
- Content of the file in the index is the same as in the destination commit (no warning is thrown)

Checkout - Alloy

```
...
let CA = (head.s).(marks.s).abs :> Blob,
    IA = s.pathcontents,
    CB = (b.marks.s).abs :> Blob
...
```

Checkout - Alloy

```
...
let CA = (head.s).(marks.s).abs :> Blob,
    IA = s.pathcontents,
    CB = (b.marks.s).abs :> Blob
...
```

```
all f:index.s | f.path -> f.blob in (IA - CA)
                => (f.path in CB.univ
                    => (f.path -> f.blob in CB or (f.path).CA = (f.path).CB)
                    else f.path not in CA.univ)
...
```

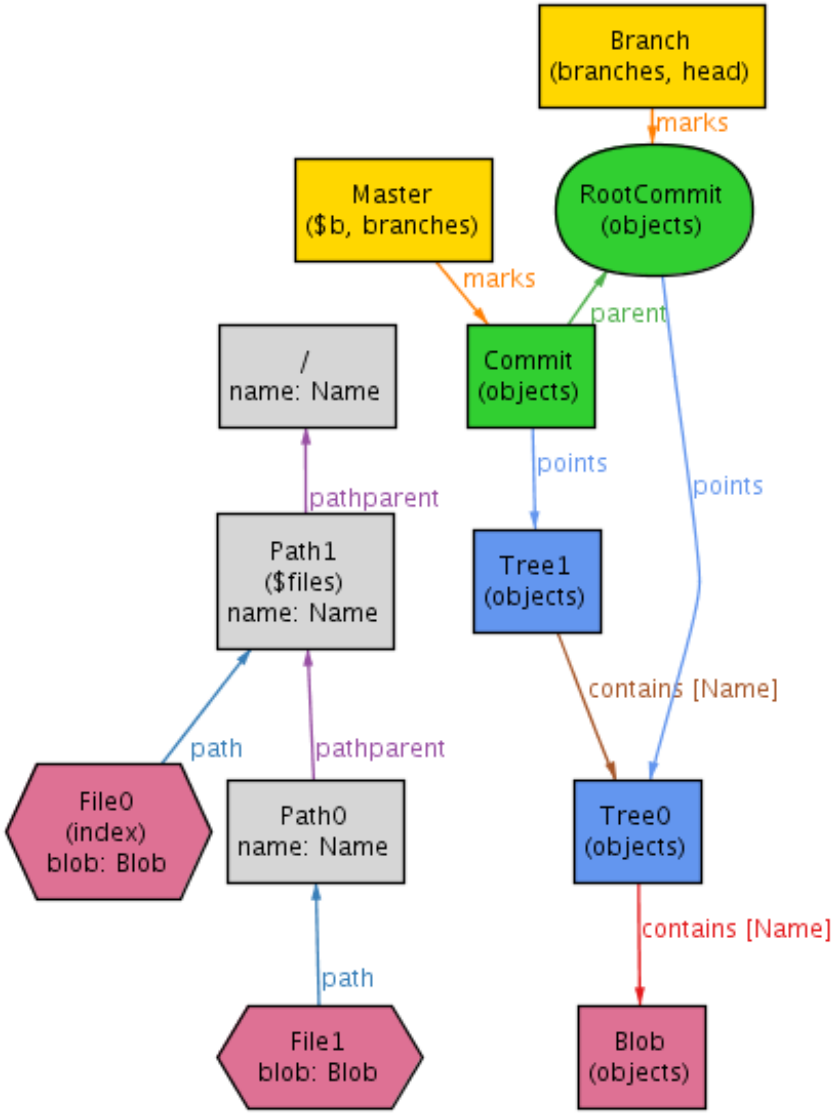
Checkout - Alloy

```
...
let CA = (head.s).(marks.s).abs :> Blob,
    IA = s.pathcontents,
    CB = (b.marks.s).abs :> Blob
...
```

```
all f:index.s | f.path -> f.blob in (IA - CA)
                => (f.path in CB.univ
                      => (f.path -> f.blob in CB or (f.path).CA = (f.path).CB)
                      else f.path not in CA.univ)
...
```

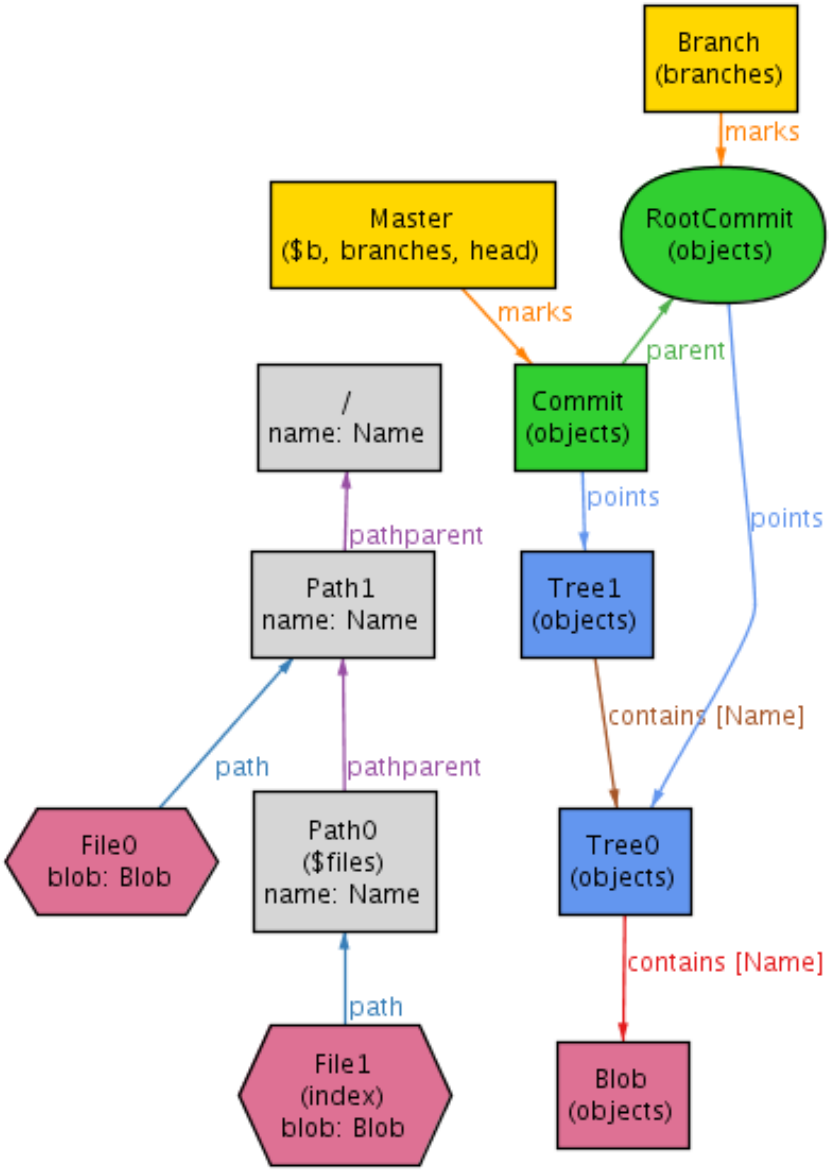
```
s'.pathcontents = CB ++ (IA - CA)
...
```

Checkout - Instance 1



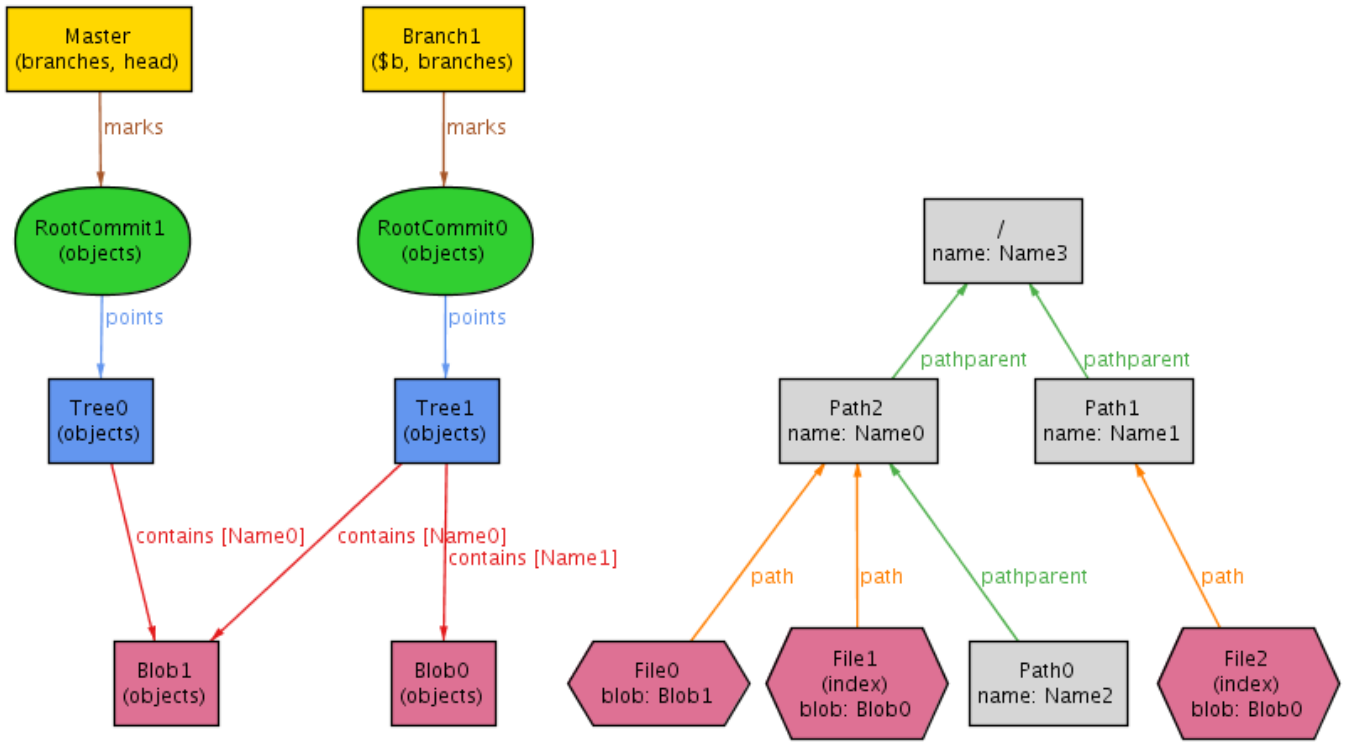
CSAIL

Checkout - Instance 1

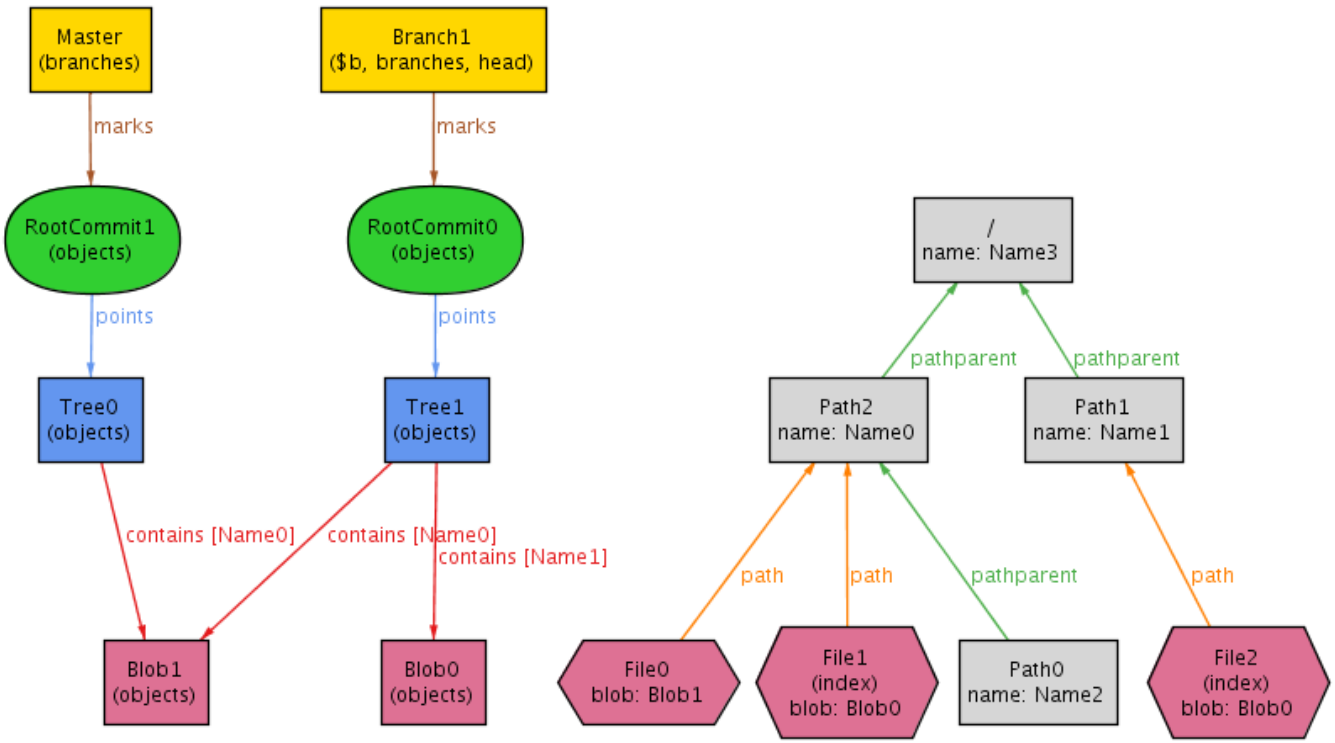


CSAIL

Checkout - Instance 2



Checkout - Instance 2



CSAIL

Checkout - Bug

- touch f

Checkout - Bug

- touch f
- git add f

Checkout - Bug

- touch f
- git add f
- git commit

Checkout - Bug

- touch f
- git add f
- git commit
- git rm f

Checkout - Bug

- touch f
- git add f
- git commit
- git rm f
- mkdir f

Checkout - Bug

- touch f
- git add f
- git commit
- git rm f
- mkdir f
- touch f/g

Checkout - Bug

- touch f
- git add f
- git commit
- git rm f
- mkdir f
- touch f/g
- git add f/g

Checkout - Bug

- touch f
- git add f
- git commit
- git rm f
- mkdir f
- touch f/g
- git add f/g
- git checkout b

Why we assume it is a bug

Why we assume it is a bug

Git Mailing list

- ”I think providing a link to that "alloy" thing could be helpful.”

Why we assume it is a bug

Git Mailing list

- "I think providing a link to that "alloy" thing could be helpful."
- "When you create a branch, it will contain everything committed on the branch you created it from at that given point. So if you commit more things on the master branch like you have done (after creating b), then switch to branch b, they won't appear. This is the correct behavior. Does that answer your question?"

Why we assume it is a bug

Git Mailing list

- "I think providing a link to that "alloy" thing could be helpful."
- "When you create a branch, it will contain everything committed on the branch you created it from at that given point. So if you commit more things on the master branch like you have done (after creating b), then switch to branch b, they won't appear. This is the correct behavior. Does that answer your question?"
- "Yes, that looks like a bug. Checkout should not overwrite uncommitted files. It does the right thing if you do not "git add f/g" (it complains that deleting the directory would lose untracked files). But if the file has been added to the index, we seem to miss the check."

Properties

Invariant preservation

All operations must preserve the invariant

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

- There is some commit iff exists at least one branch and an head
- The current branch must exist and must have a commit
- All objects from one state descend from one of its commits
- Referential integrity is kept on dynamic relations
- There are no empty trees

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
```

Properties

Checkout, all files from commit will be in index

When a checkout is performed, all files that are on the commit pointed by b, will be in the index.

```
all s,s':State, b:branches.s, p:Path, blo:Blob |
      p->blo in (b.marks.s).abs => some f:File |
         f.path = p and f.blob = blo and f inindex.s'
```

Properties

Checkout, all files from commit will be in index

When a checkout is performed, all files that are on the commit pointed by b, will be in the index.

```
all s,s':State, b:branches.s, p:Path, blo:Blob |
      p->blo in (b.marks.s).abs => some f:File |
         f.path = p and f.blob = blo and f inindex.s'
```

Counter Examples found

Future work

- Finish the Merge operation (used for Push and Pull)
- 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 13, 2012