INCREASING MODELLING CONFIDENCE WITH UNSAT CORE

Daniel Jackson · SPIN Workshop · Los Angeles · Aug 12, 2008

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

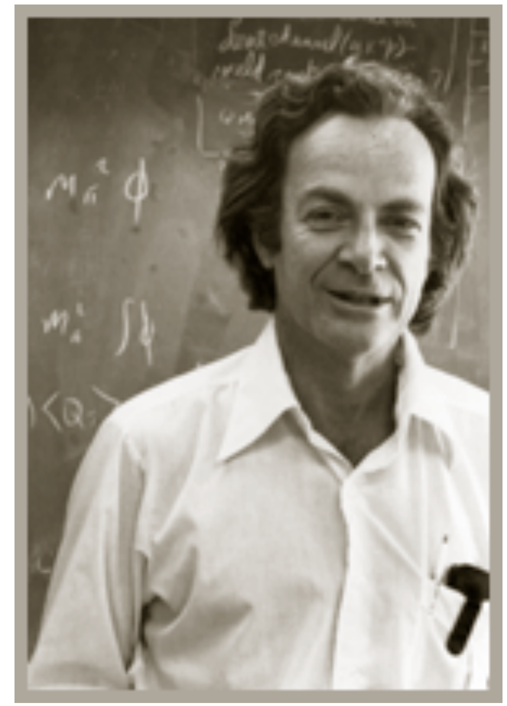
design models

my premise

- design determines software quality
- especially conceptual structure

design verification

- in hardware: catch subtle bugs
- in software: keep designer honest

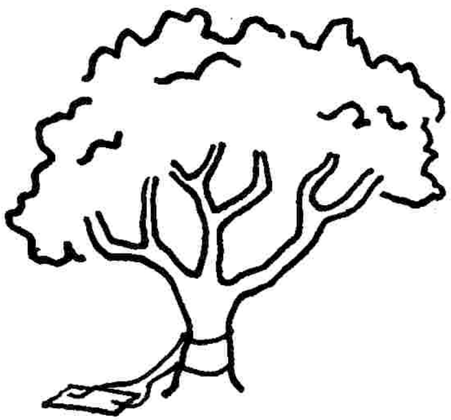


The first principle is that
you must not fool yourself, and
you are the easiest person to fool

Richard Feynman

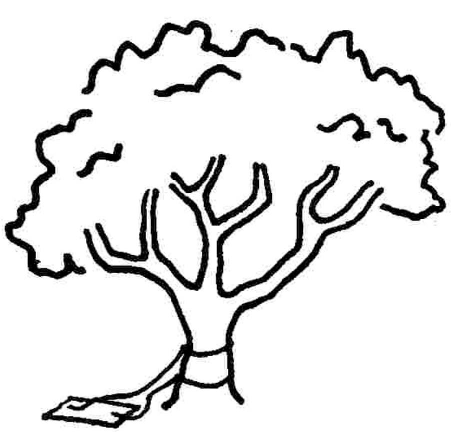
wishful thinking

wishful thinking

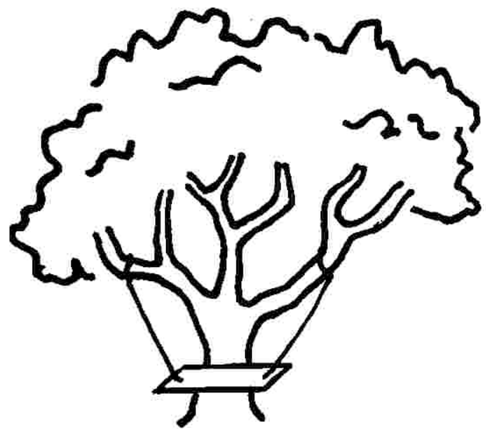


as designed

wishful thinking

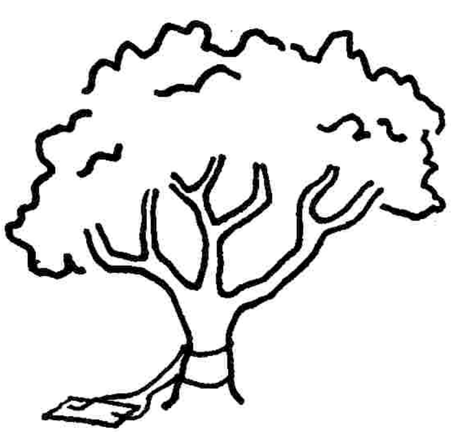


as designed

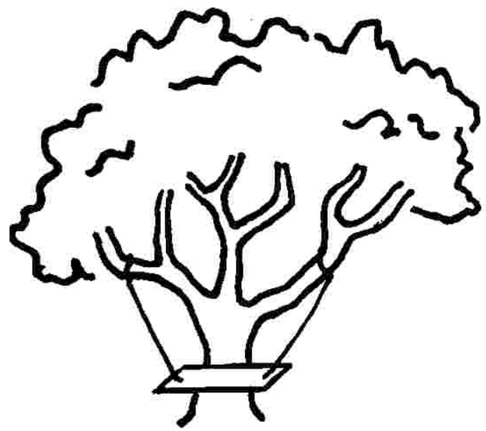


as built

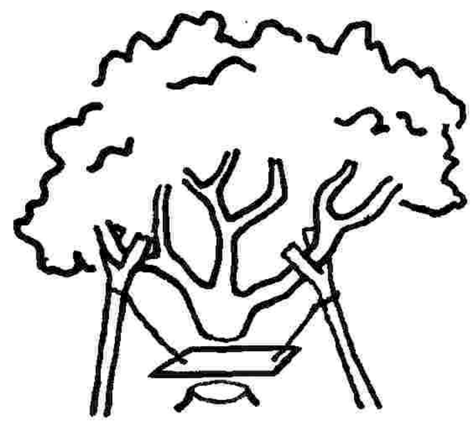
wishful thinking



as designed



as built



as installed

wishful thinking

as designed as built as installed as wanted

wishful thinking

as designed   as built   as installed   as wanted

how formalization helps

- forces clear expression
- exposes inconsistency

wishful thinking

as designed as built as installed as wanted

how formalization helps

- forces clear expression
- exposes inconsistency

but risks remain

- model may not match designer's intent
- checks may not cover the model

a mixed blessing

declarative modelling

- describe by properties, not mechanism
- can make models more direct and succinct
- but raises risk of inconsistency

```
sig State {
   undoAction: State -> State
   }

pred do (s, s': State) {
   s'.undoAction [s'] = s
   }

pred undo (s, s': State) {
   s' = s.undoAction [s]
   }

check {
   all s, s', s": State | do [s, s'] and undo [s', s"] implies s = s"
   }
```

examples

ex1: overconstraint

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
  outcome =
    { n: Node, v: Value |
      let sn = { x: Node | all y: Node | x.id < y.id } |
        v = sn.vote
    }
  }

assert consensus {
  one v: Value | all n: Node | n.outcome = v
  some n: Node | n.outcome = n.vote
  }

check consensus for 3
```

```
Executing "Check consensus for 3"
  Solver=minisat(jni) Bitwidth=4 MaxSeq=3 SkolemDepth=2 Symmetry=20
  1153 vars. 72 primary vars. 3059 clauses. 29ms.
  No counterexample found. Assertion may be valid. 2ms.
```

ex1: overconstraint

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node | x.id < y.id } |
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }

check consensus for 3
```

```
Executing "Check consensus for 3"
  Solver=minisat(jni) Bitwidth=4 MaxSeq=3 SkolemDepth=2 Symmetry=20
  1153 vars. 72 primary vars. 3059 clauses. 29ms.
  No counterexample found. Assertion may be valid. 2ms.
```

fixing the model

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
   outcome =
      { n: Node, v: Value |
         let sn = { x: Node | all y: Node - x | x.id < y.id } |
            v = sn.vote
      }
   }

assert consensus {
   one v: Value | all n: Node | n.outcome = v
   some n: Node | n.outcome = n.vote
   }

check consensus for 3
```

```
Executing "Check consensus for 3"
  Solver=minisat(jni) Bitwidth=4 MaxSeq=3 SkolemDepth=2 Symmetry=20
  1153 vars. 72 primary vars. 3059 clauses. 40ms.
  No counterexample found. Assertion may be valid. 2ms.
```

ex2: weak check

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
    objects: set Object,
    links: Alias -> Object,
    contents: Dir -> Object
    }

abstract sig Action {
    pre, post: FileSystem
    }

sig DeleteAction extends Action {
    obj: Object
    } {
    post.objects = pre.objects - obj
    post.contents = pre.contents - obj->Object - Object->obj
    post.links = pre.links - Alias->obj - obj->Alias
    }

check {
    all d: DeleteAction | d.obj not in d.post.objects
    }
```

```
Executing "Check check$1"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  1893 vars. 108 primary vars. 4058 clauses. 38ms.
  No counterexample found. Assertion may be valid. 27ms.
```

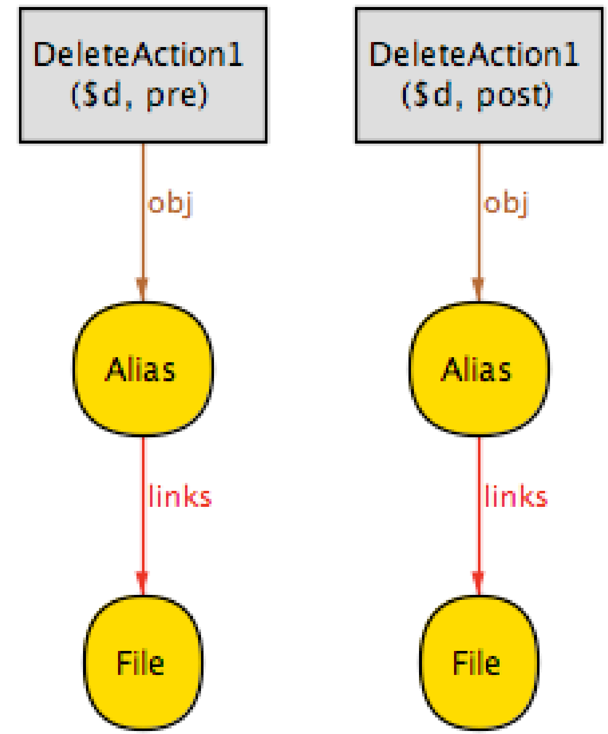
ex2: weak check

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
   objects: set Object,
   links: Alias -> Object,
   contents: Dir -> Object
   }

abstract sig Action {
   pre, post: FileSystem
   }

sig DeleteAction extends Action {
   obj: Object
   } {
   post.objects = pre.objects - obj
   post.contents = pre.contents - obj->Object - Object->obj
   post.links = pre.links - Alias->obj - obj->Alias
   }

check {
   all d: DeleteAction | d.obj not in d.post.objects
   }
```

```
Executing "Check check$1"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  1893 vars. 108 primary vars. 4058 clauses. 38ms.
  No counterexample found. Assertion may be valid. 27ms.
```

stronger check finds bug

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
   objects: set Object,
   links: Alias -> Object,
   contents: Dir -> Object
   }

abstract sig Action {
   pre, post: FileSystem
   }

sig DeleteAction extends Action {
   obj: Object
   } {
   post.objects = pre.objects - obj
   post.contents = pre.contents - obj->Object - Object->obj
   post.links = pre.links - Alias->obj - obj->Alias
   }

pred inv (fs: FileSystem) {
   fs.(links+contents).Object + Object.(fs.(contents+links)) in fs.objects
   }

check {
   all d: DeleteAction | d.pre.inv implies d.post.inv
   }
```



ex3: scope too small

```
sig Time {
    members: set Member
    }
sig Member {
    friend: Member -> Time
    }
abstract sig Event {
    pre, post: Time,
    m: Member
    }
sig Join extends Event {} {
    post.members = pre.members + m
    nochange [friend]
    }
sig Befriend extends Event {f: Member} {
    m + f in pre.members
    friend.post = friend.pre + m->f
    nochange [members]
    }

fact init {
    no first.members
    no friend.first
    }

assert Symmetric {
    all t: Time | friend.t = ~(friend.t)
    }
check Symmetric for 2
```

```
Executing "Check Symmetric for 2"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=2 Skolem
  513 vars. 44 primary vars. 933 clauses. 17ms.
  No counterexample found. Assertion may be valid. 6ms.
```

ex3: scope too small

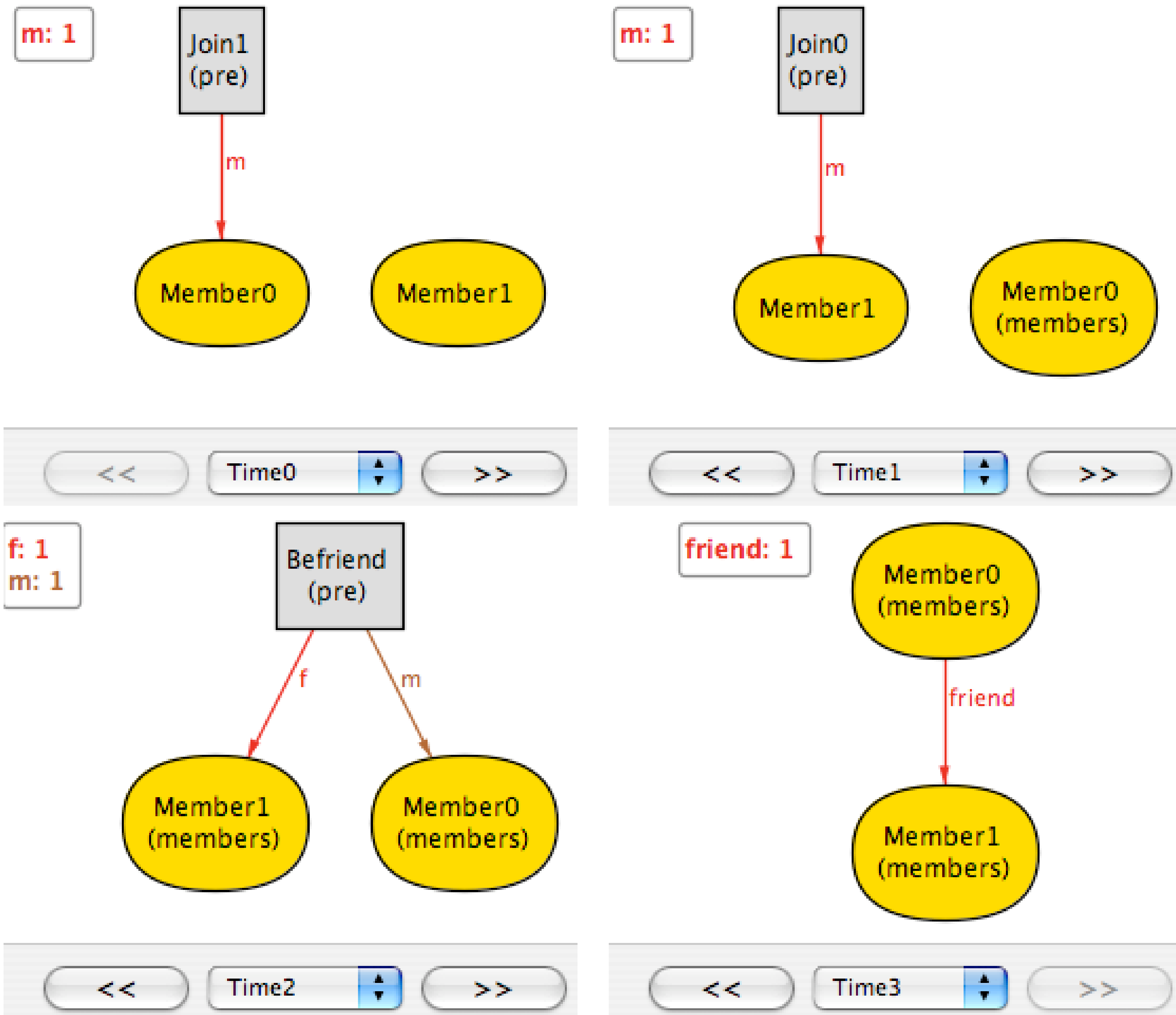
```
sig Time {
    members: set Member
    }
sig Member {
    friend: Member -> Time
    }
abstract sig Event {
    pre, post: Time,
    m: Member
    }
sig Join extends Event {} {
    post.members = pre.members + m
    nochange [friend]
    }
sig Befriend extends Event {f: Member} {
    m + f in pre.members
    friend.post = friend.pre + m->f
    nochange [members]
    }

fact init {
    no first.members
    no friend.first
    }

assert Symmetric {
    all t: Time | friend.t = ~(friend.t)
    }
check Symmetric for 2
```

```
Executing "Check Symmetric for 2"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=2 Skolem
  513 vars. 44 primary vars. 933 clauses. 17ms.
  No counterexample found. Assertion may be valid. 6ms.
```

scope was too small

```
sig Time {
   members: set Member
   }
sig Member {
   friend: Member -> Time
   }
abstract sig Event {
   pre, post: Time,
   m: Member
   }
sig Join extends Event {} {
   post.members = pre.members + m
   nochange [friend]
   }
sig Befriend extends Event {f: Member} {
   m + f in pre.members
   friend.post = friend.pre + m->f
   nochange [members]
   }

fact init {
   no first.members
   no friend.first
   }

assert Symmetric {
   all t: Time | friend.t = ~(friend.t)
   }
check Symmetric for 4
```

counterexample



approaches

how can we mitigate this risk?

- avoid these problems
- increase confidence

approach #1: build it and try it

- do this anyway, tantamount to giving up

approach #2: simulation

- run sample scenarios
- a good approach

ex1 revisited

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node | x.id < y.id } |
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }
```

ex1 revisited

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node | x.id < y.id } |
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }

run {}
```

ex1 revisited

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node | x.id < y.id } |
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }

run {}
```

```
Executing "Run run$1"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  1024 vars. 72 primary vars. 862 clauses. 31ms.
  No instance found. Predicate may be inconsistent. 3ms.
```

simulation deficiencies

but simulation can't expose the other problems

scope too small

- simulations succeed for Join and Befriend events
- need two Joins and a Befriend to catch the flaw

weak check

- model is not overconstrained and check is not vacuous
- problem is that check doesn't cover model

what’s the problem

counterexamples are good

- counterexample -> information
- no counterexample -> no information?

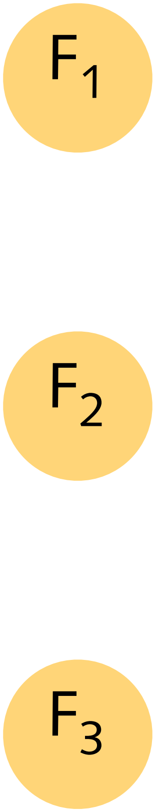
what information can we get from an inconsistency?

idea: show user the <u>proof</u>

- SAT solver is a theorem prover
- when no counterexample, generates proof
- translate back into source language?

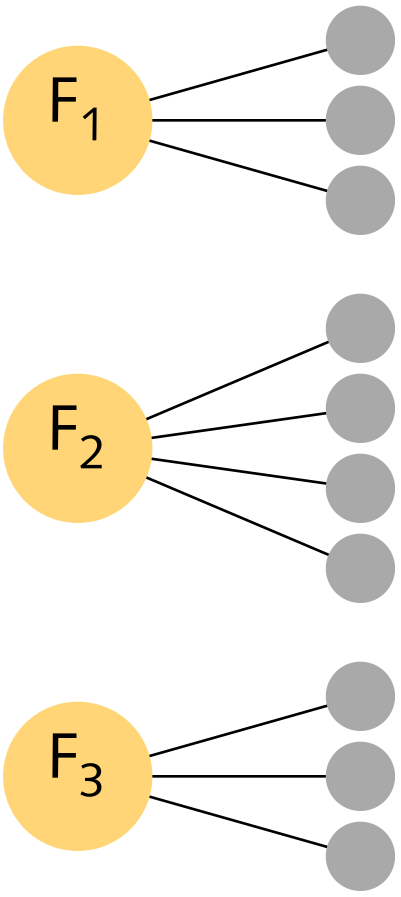
exploiting the proof

- source formulas (Alloy)



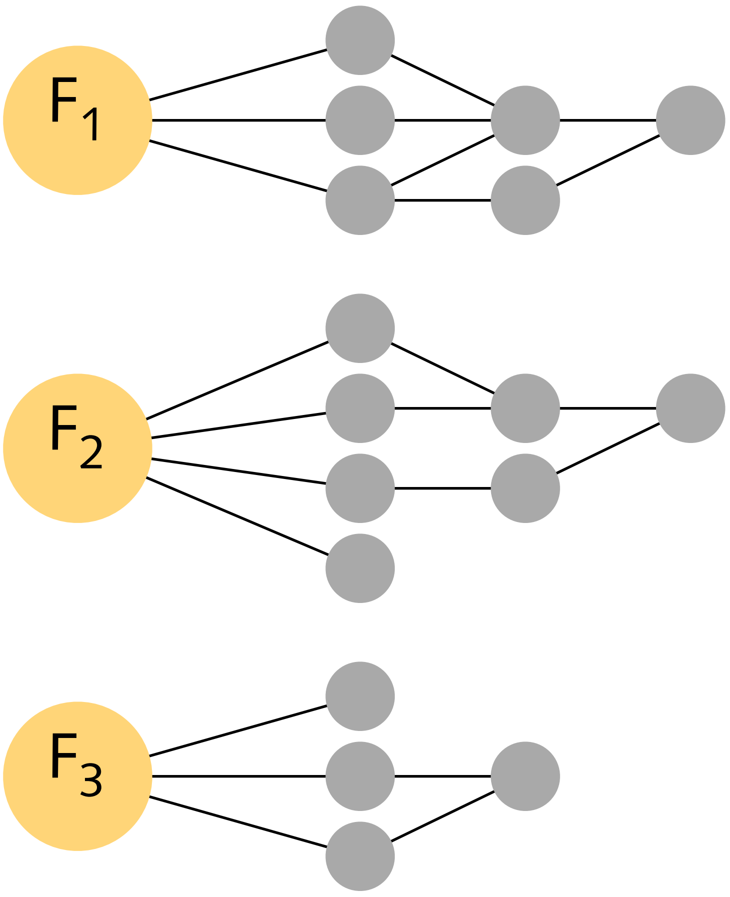
exploiting the proof

- source formulas (Alloy)
- translated to CNF (SAT)



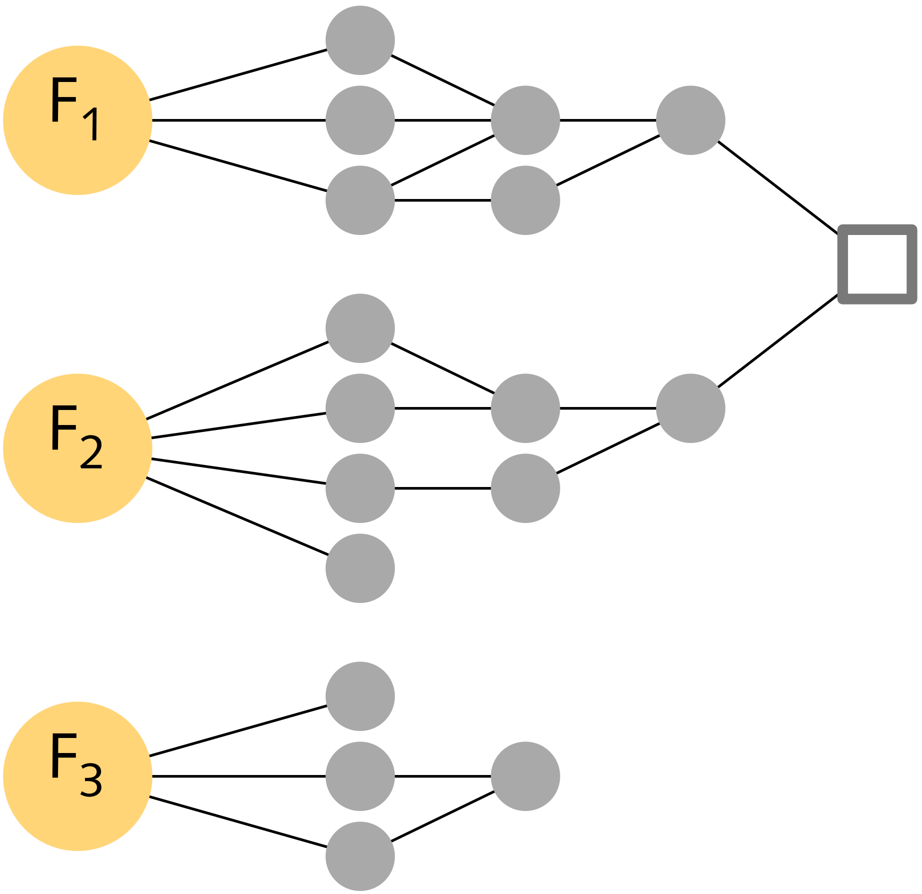
exploiting the proof

- source formulas (Alloy)
- translated to CNF (SAT)
- new clauses learned



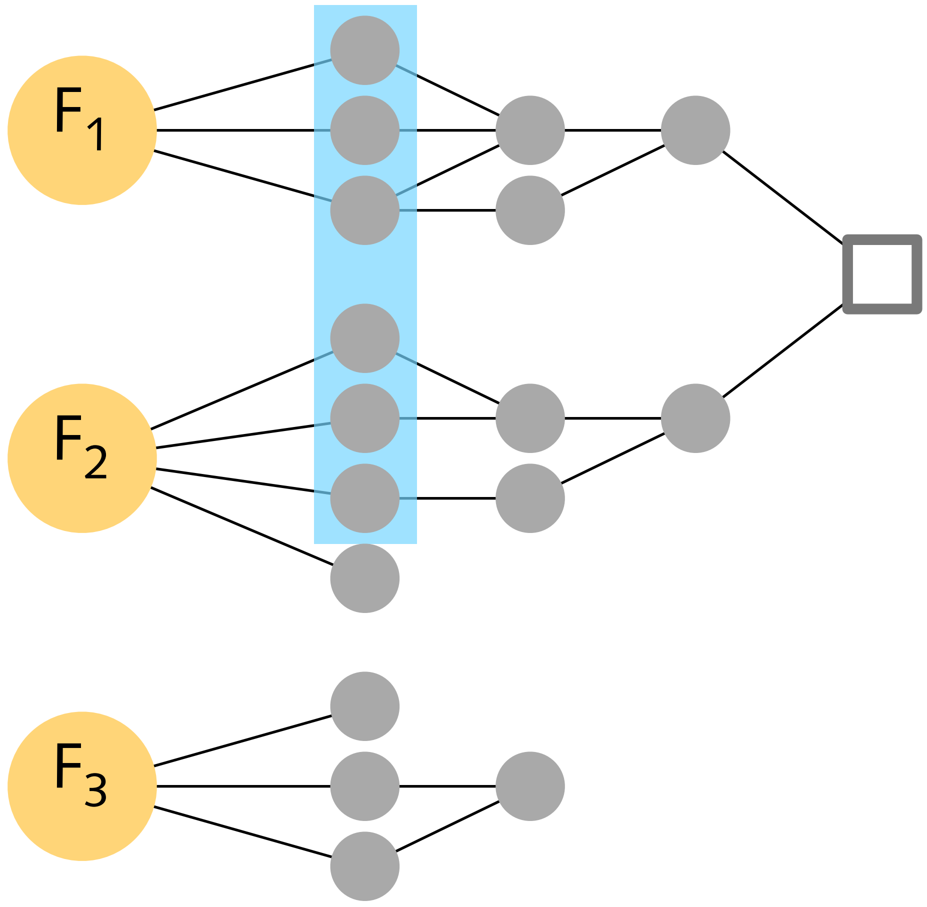
exploiting the proof

- source formulas (Alloy)
- translated to CNF (SAT)
- new clauses learned
- eventually, conflict inferred



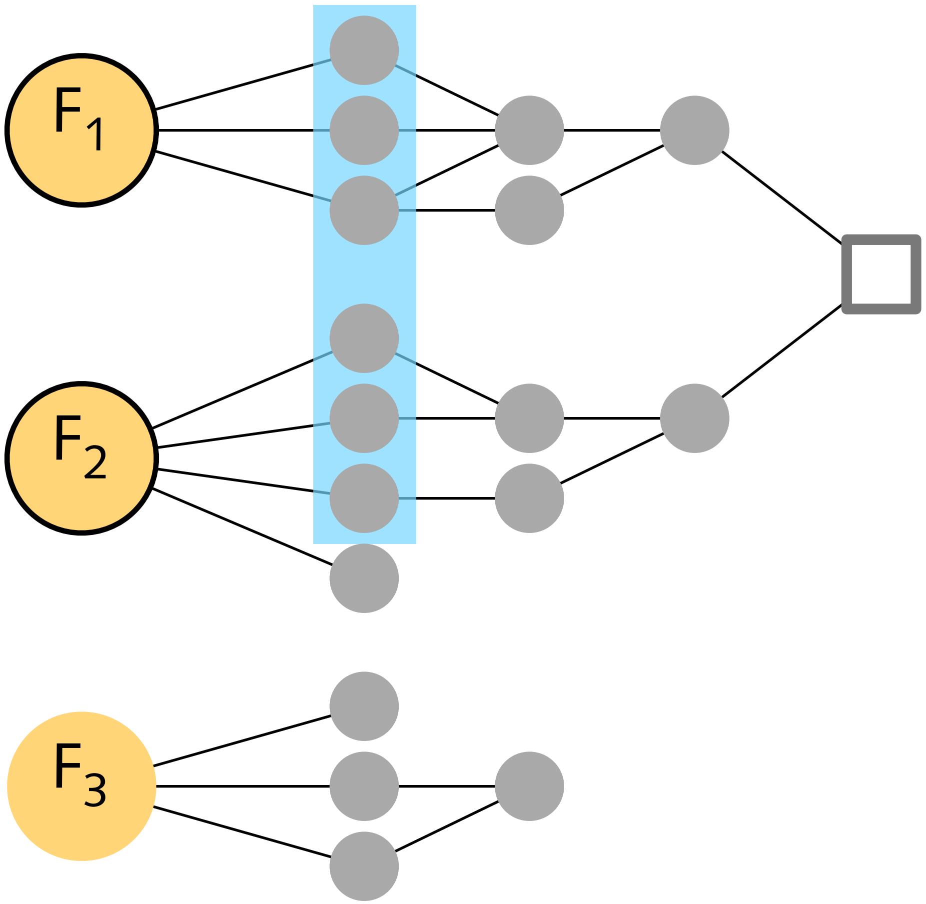
exploiting the proof

- source formulas (Alloy)
- translated to CNF (SAT)
- new clauses learned
- eventually, conflict inferred
- unsat core identified



exploiting the proof

- source formulas (Alloy)
- translated to CNF (SAT)
- new clauses learned
- eventually, conflict inferred
- unsat core identified
- source formulas marked



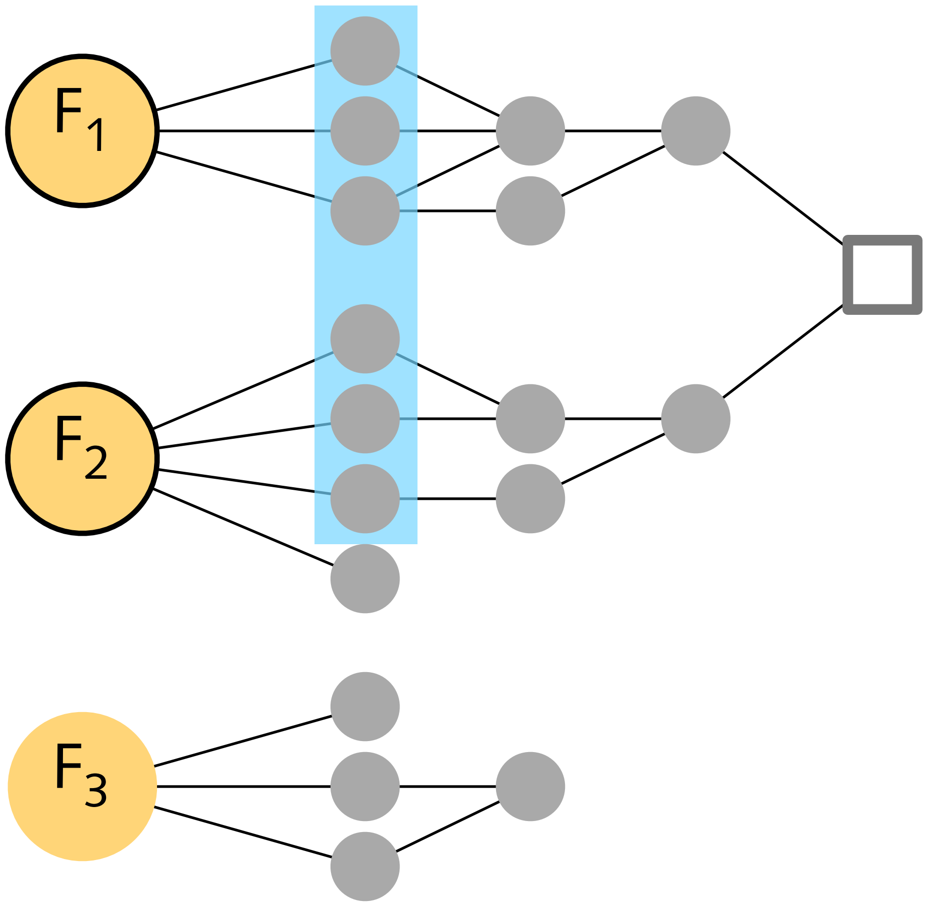
challenges

performance

- still interactive, like solving

minimality

- the smaller, the better
- minimal wrt source, not SAT
- local minimum good enough?



standard cases

model: $M_1 \wedge M_2 \wedge M_3 \ldots$

property: $P_1 \vee P_2 \vee P_3 \ldots$

obtain formula

formula: $M_1 \wedge M_2 \wedge M_3 \wedge \neg P_1 \wedge \neg P_2 \wedge \neg P_3 \ldots$

solve & extract core

core: $M_1 \wedge M_2 \wedge M_3 \wedge \neg P_1 \wedge \neg P_2 \wedge \neg P_3 \ldots$

model overconstrained

core: $M_1 \wedge M_2 \wedge M_3 \wedge \neg P_1 \wedge \neg P_2 \wedge \neg P_3 \ldots$

property vacuous

core: $M_1 \wedge M_2 \wedge M_3 \wedge \neg P_1 \wedge \neg P_2 \wedge \neg P_3 \ldots$

property too weak

some history

vacuity detection in temporal logic, 1997

- specialized algorithms for detecting vacuous checks
- Beer et al 1997; Kupferman & Vardi 1999; Chockler et al 2002; etc

unsatisfiable cores, 2003

- first developed for certifying SAT solvers
- solver generates proof as byproduct of solving
- unsat core for Chaff (Zhang & Malik) and Berkmin (Goldberg & Novikov)
- used in model checking by McMillan

history, continued

Ilya Shlyakhter et al, 2004

- added unsat core to Alloy Analyzer 3
- Chaff to get boolean core, mapped to Alloy
- but minimal Chaff core ≠ minimal Alloy core
- so resulting cores too big to be useful

Chechik, Devereux and Gurfinkel, 2004

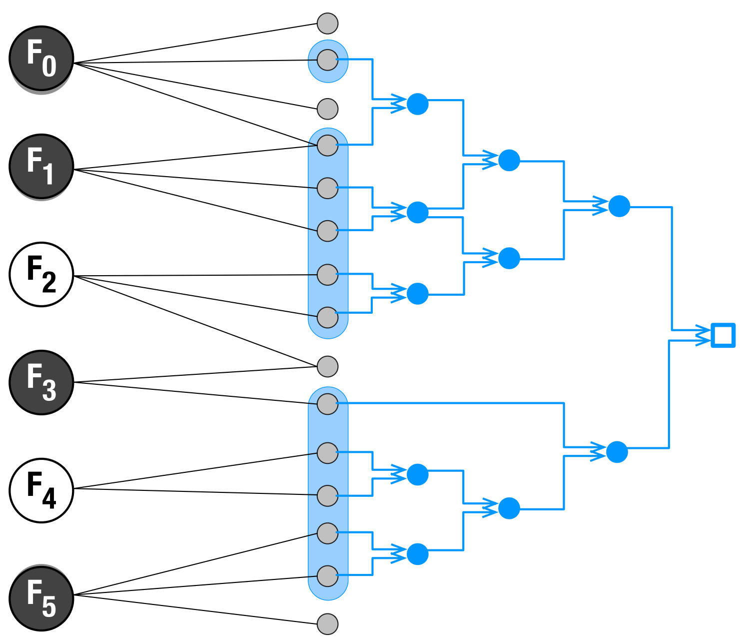
- used structure of proof to refine relevance of variables

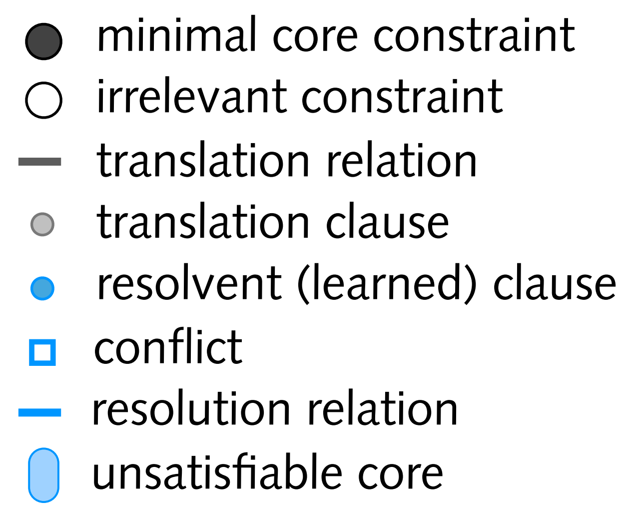
Emina Torlak, 2007

- new algorithm combines forwards and backwards methods
- guarantees minimal core at source level in reasonable time

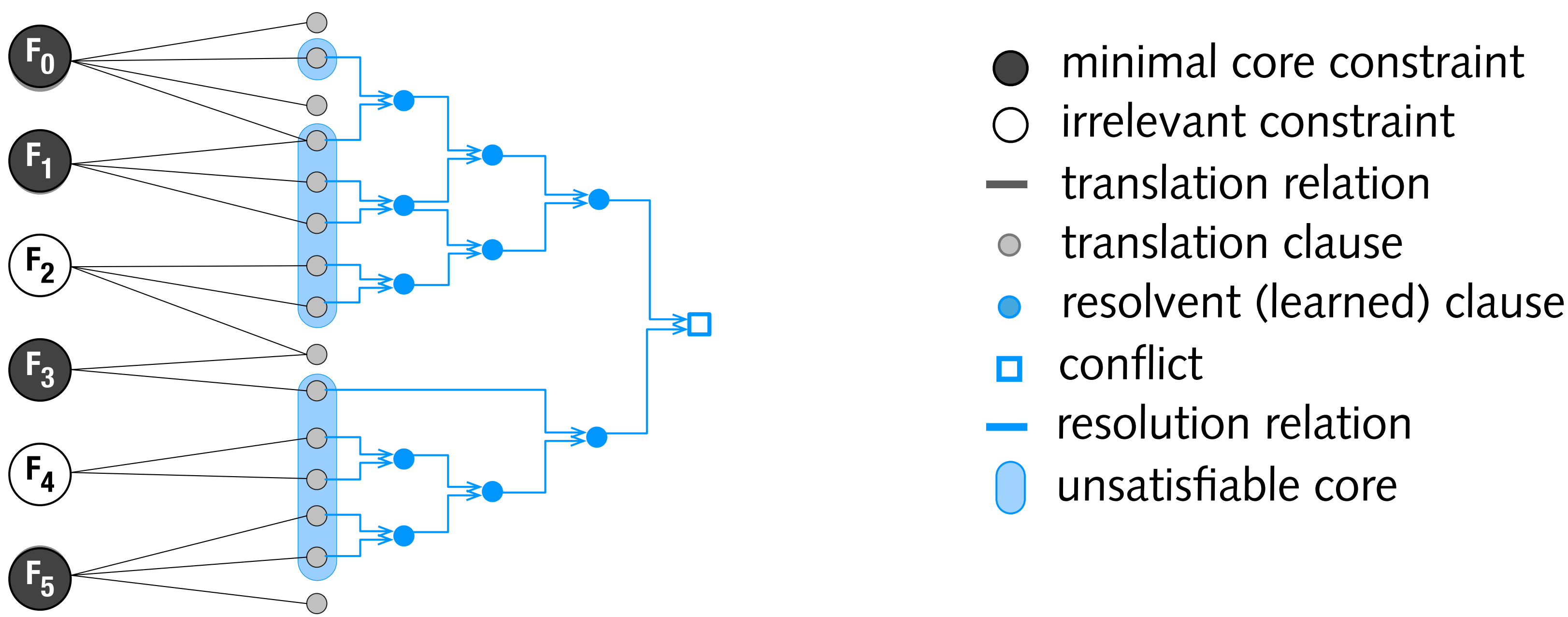
torlak's algorithm

a resolution-based framework





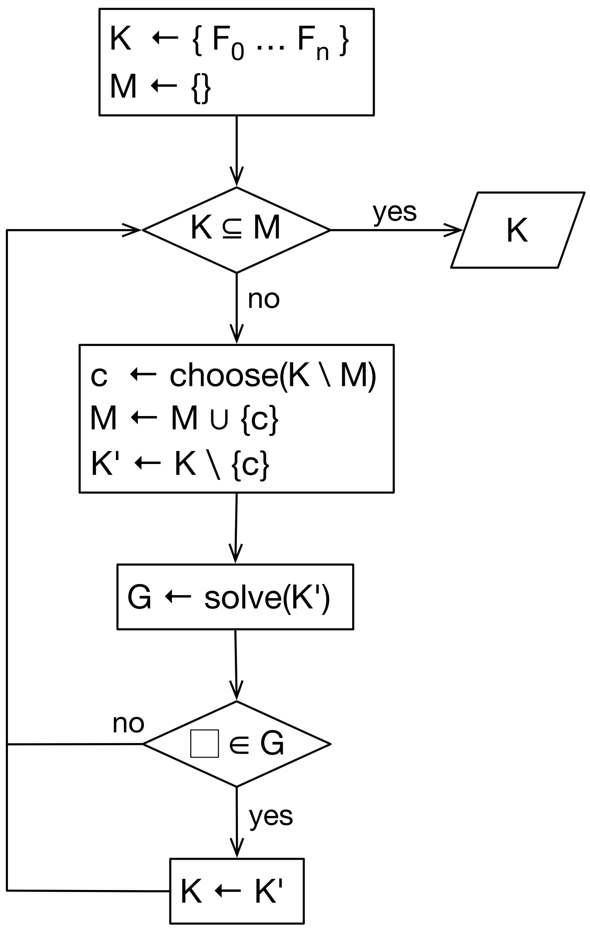
a resolution-based framework

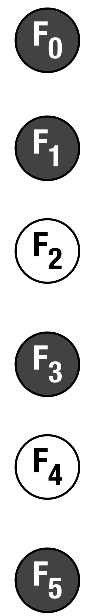


challenge

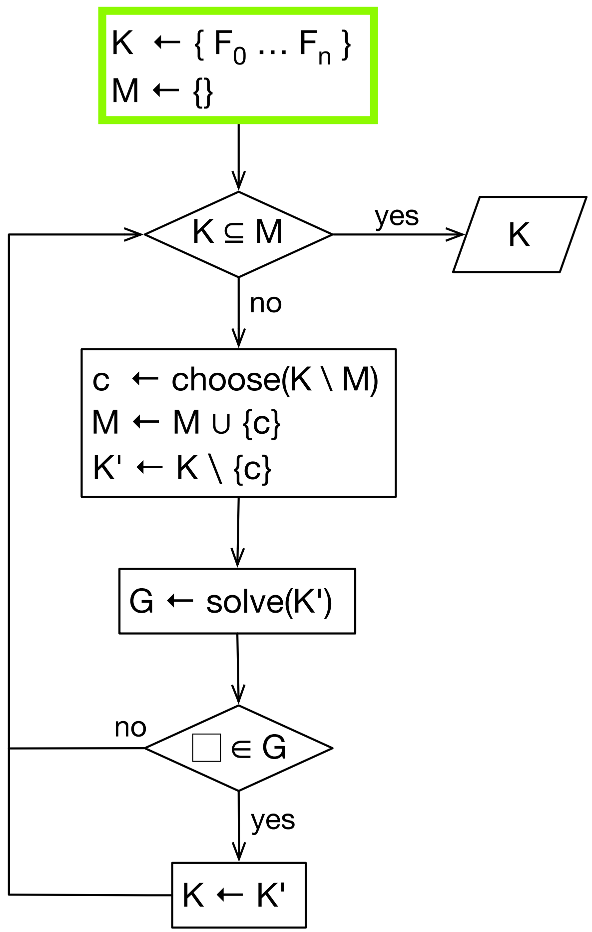
- use proof at clause level to find minimal core at source level
- even though clause proof is not minimal
- and small clause proof may map to large source proof

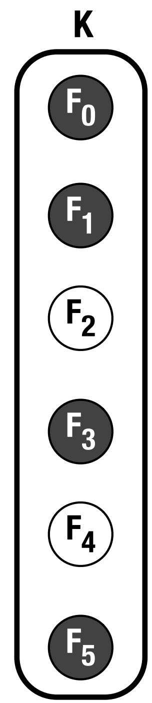
naive core extraction (NCE)



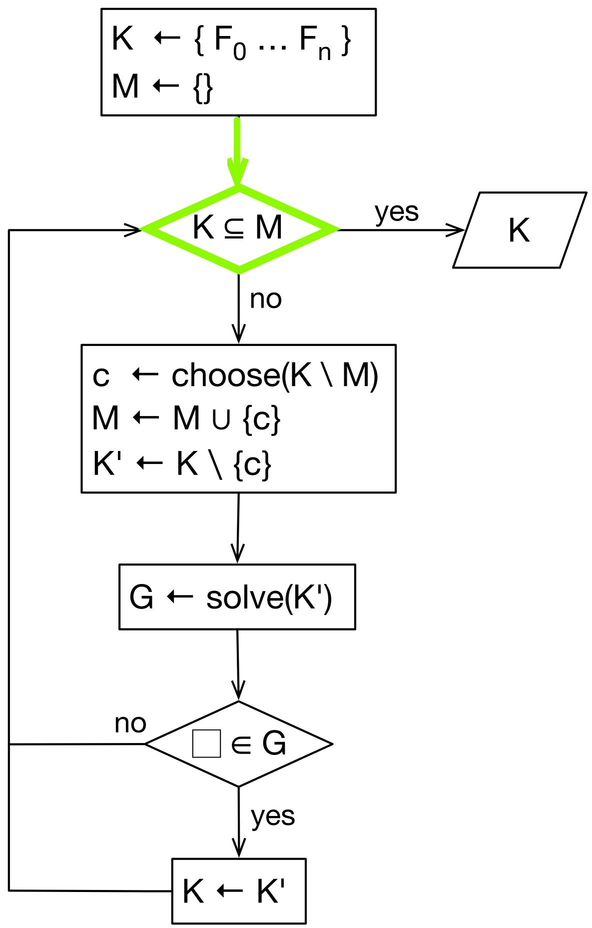


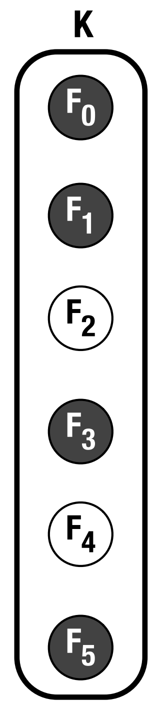
naive core extraction (NCE)



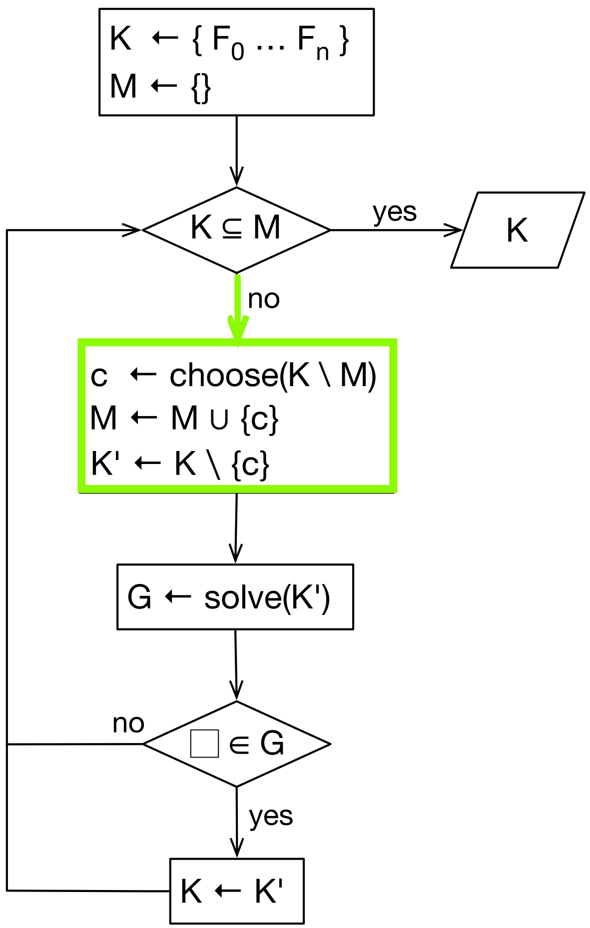


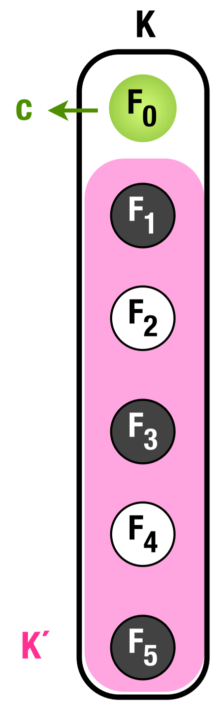
naive core extraction (NCE)



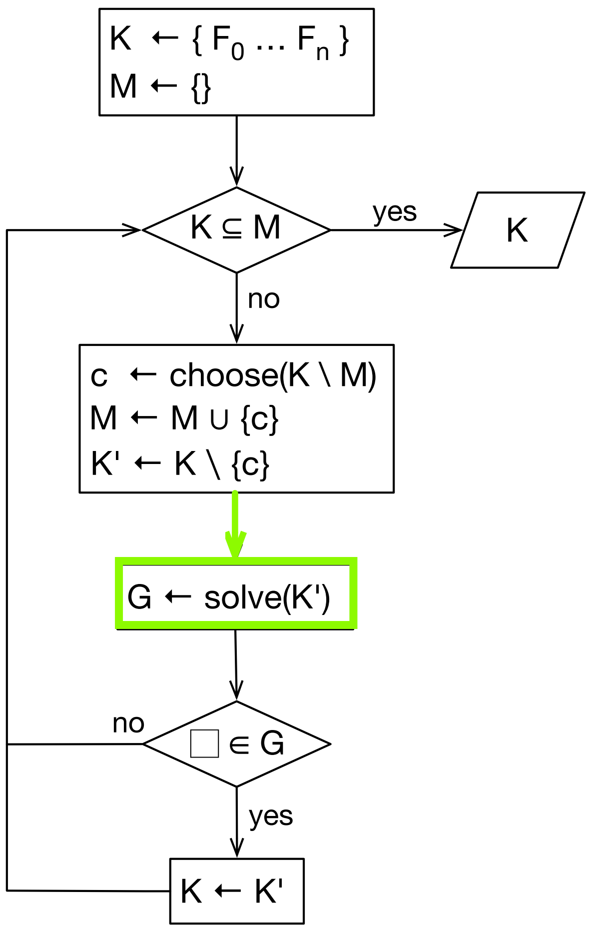


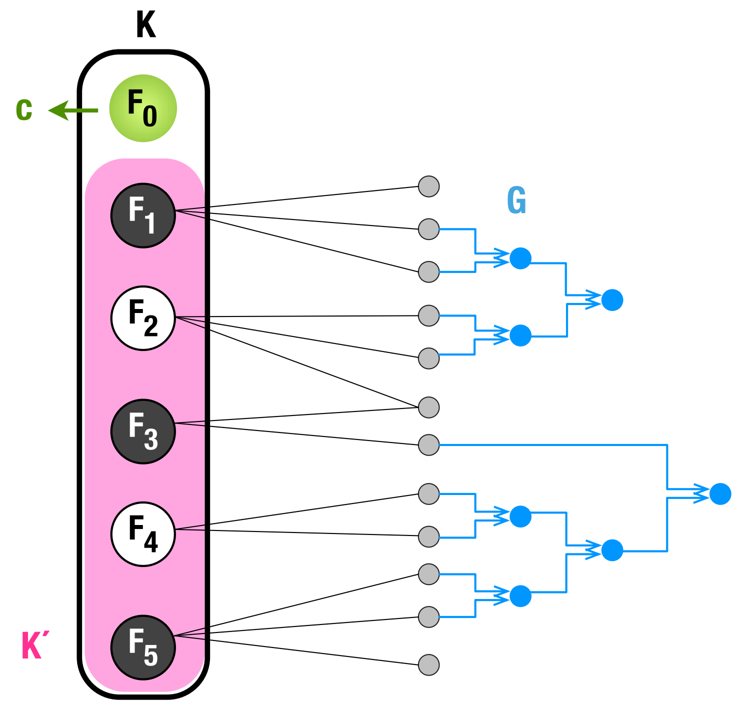
naive core extraction (NCE)



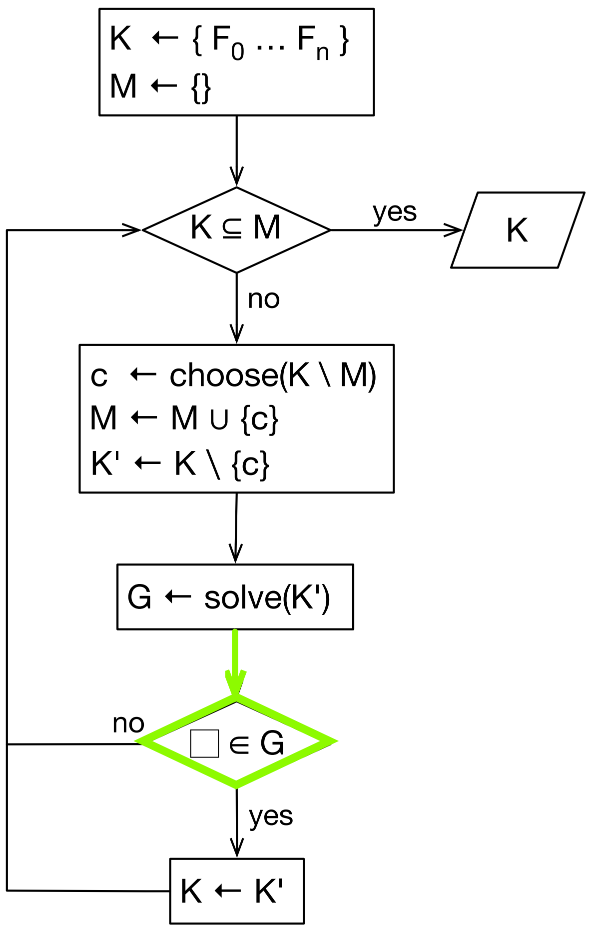


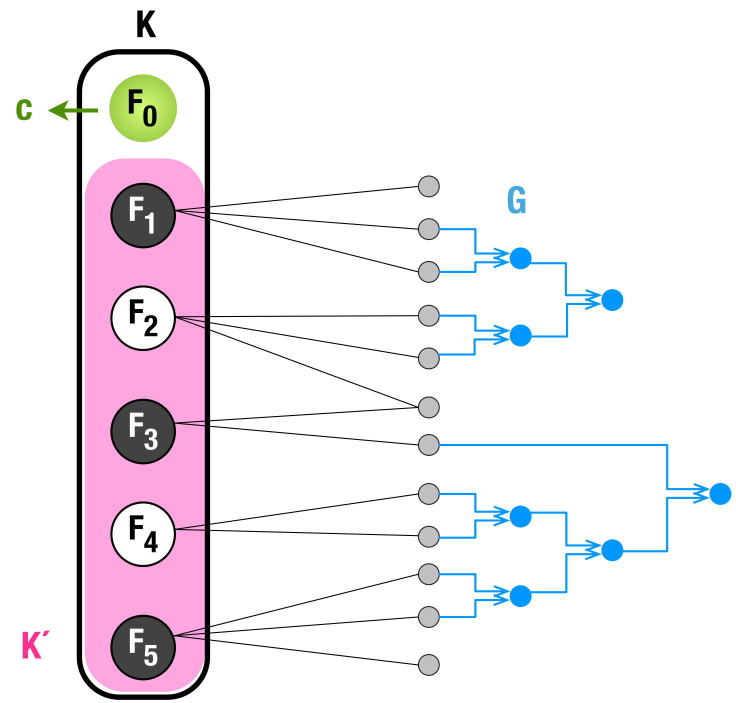
naive core extraction (NCE)



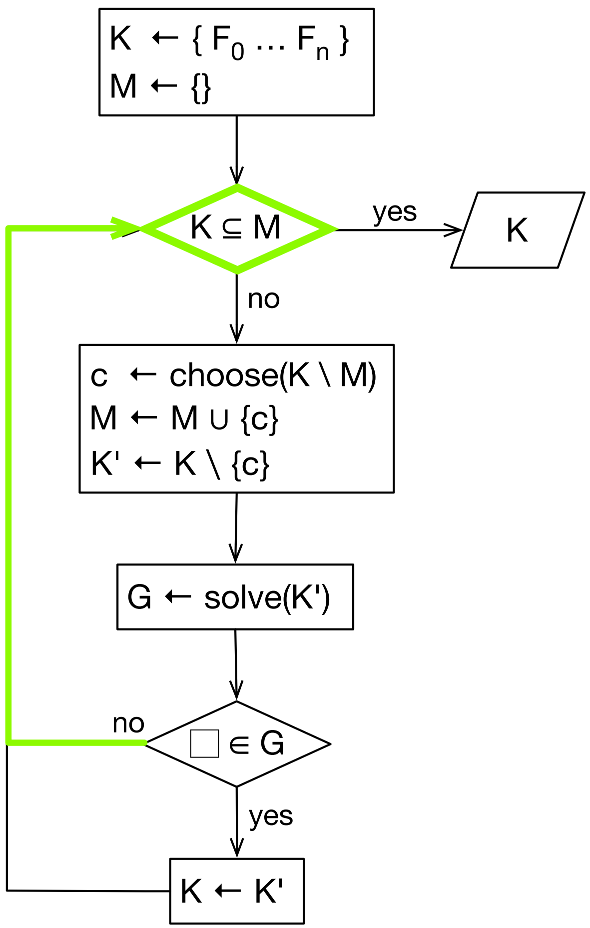


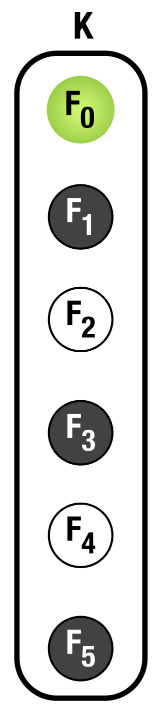
naive core extraction (NCE)



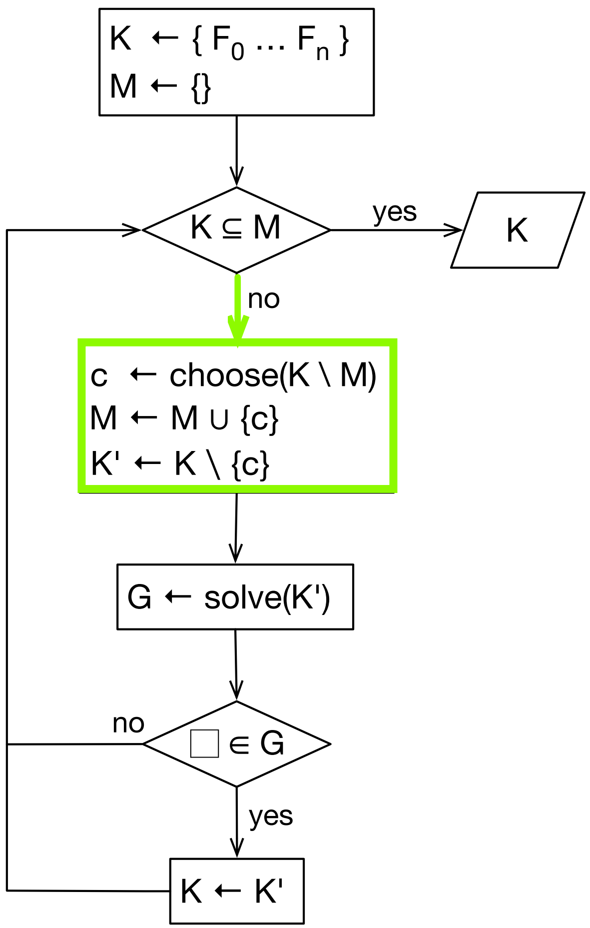


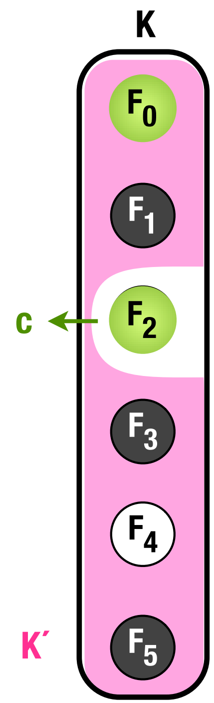
naive core extraction (NCE)



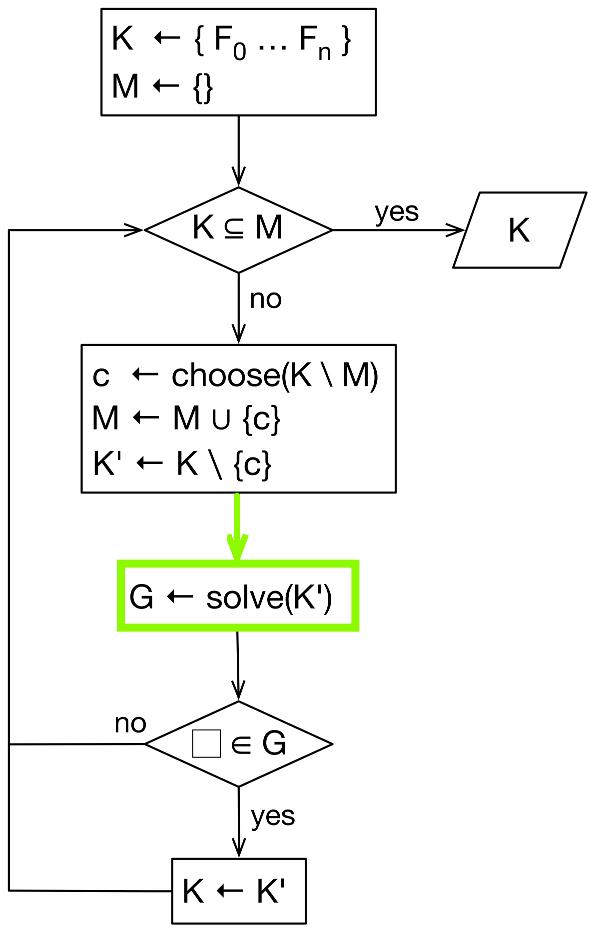


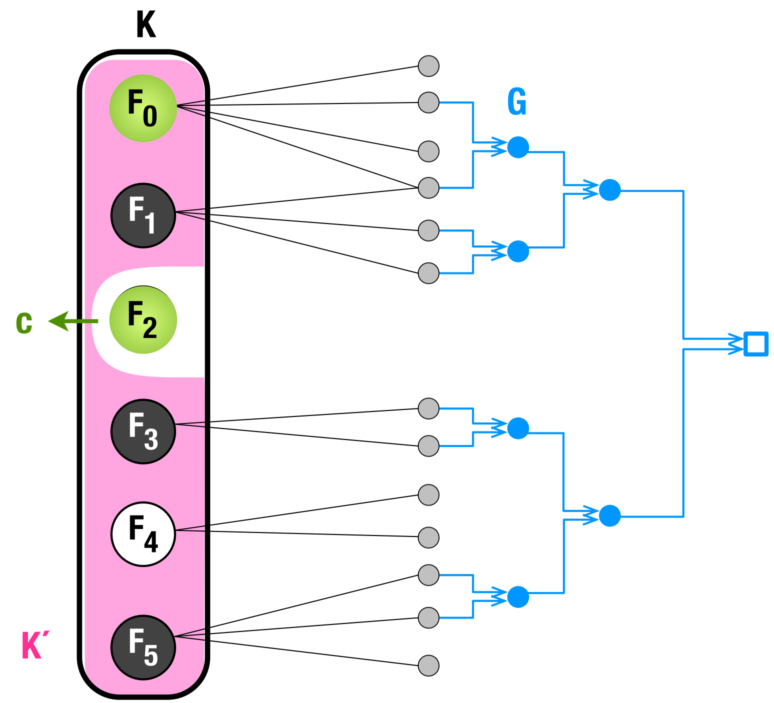
naive core extraction (NCE)



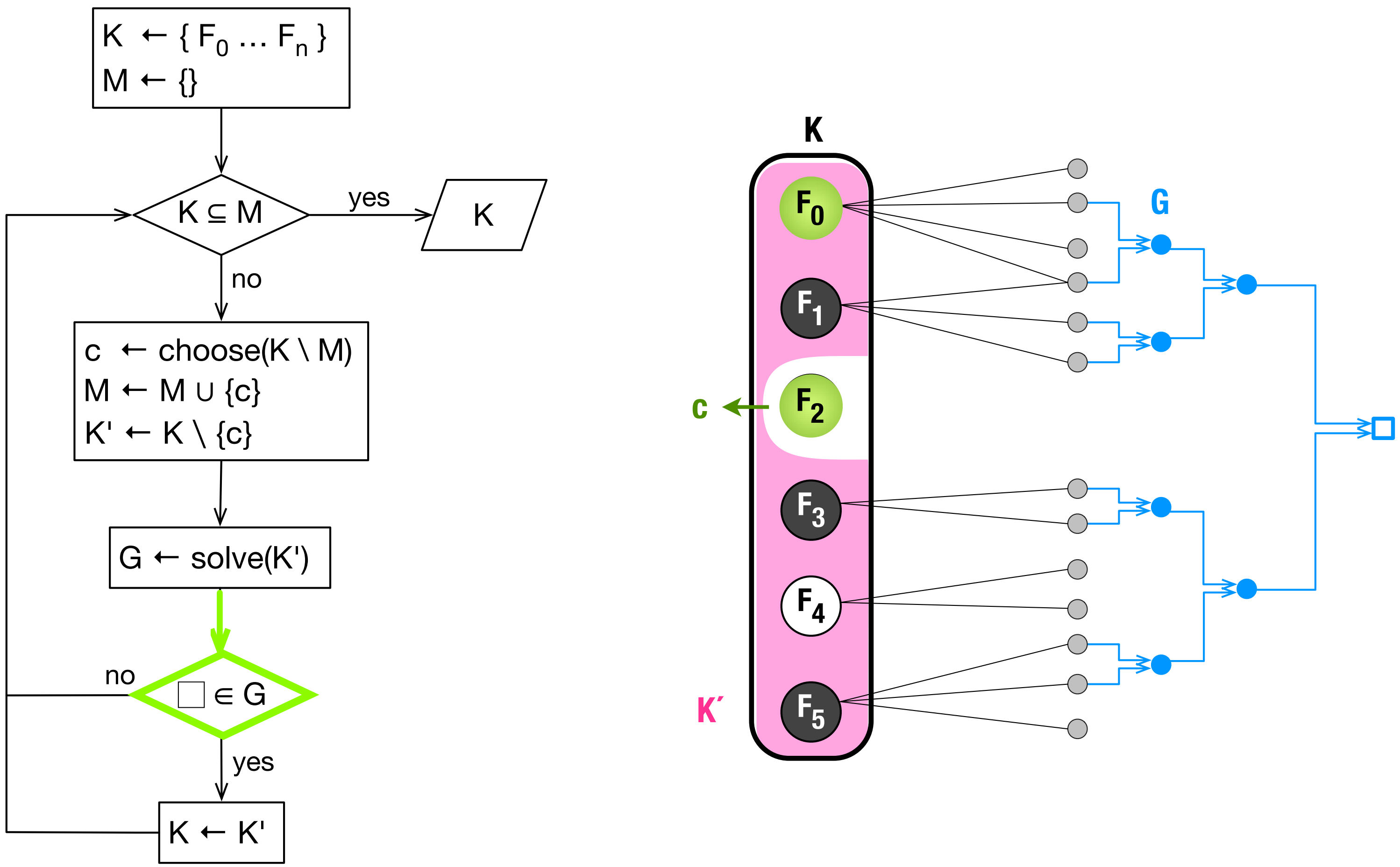


naive core extraction (NCE)

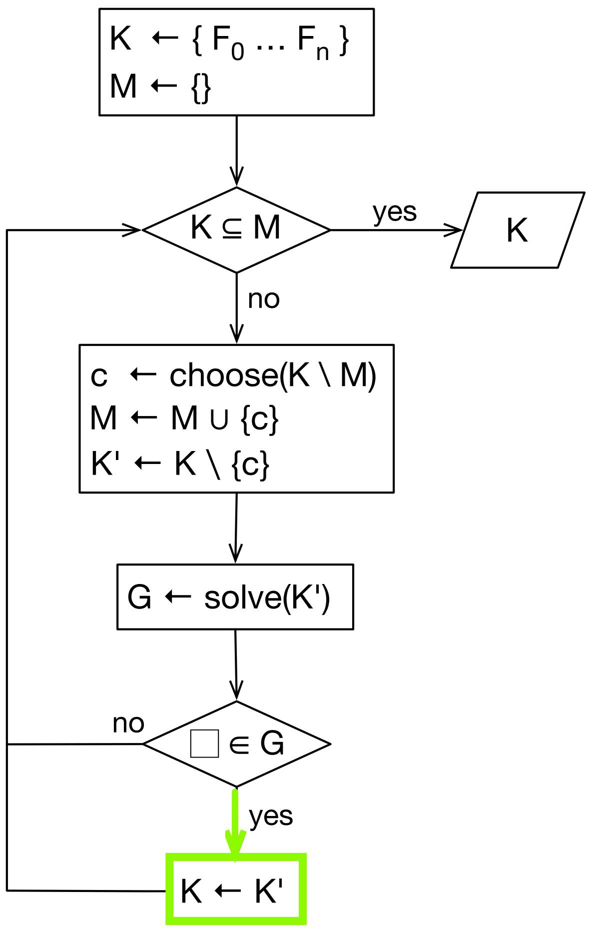


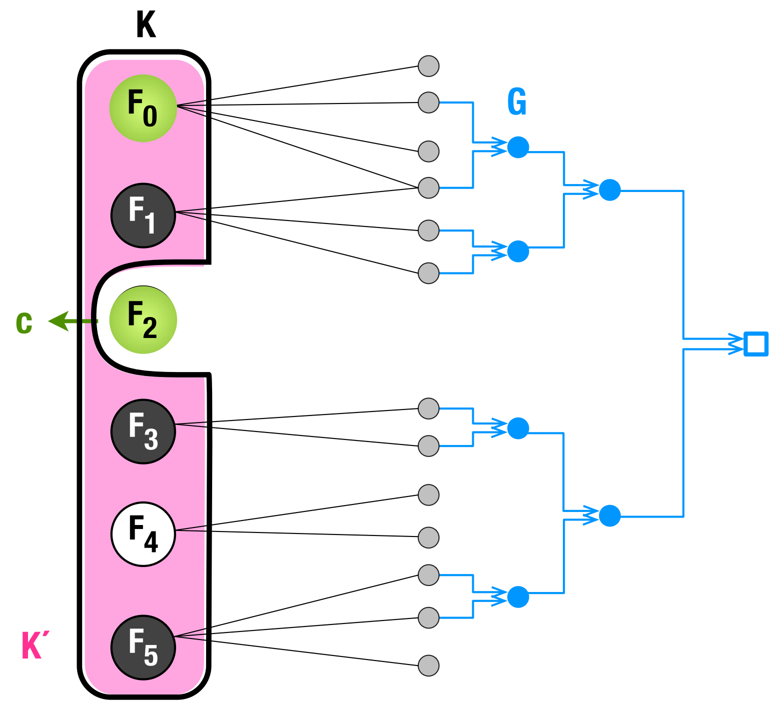


naive core extraction (NCE)

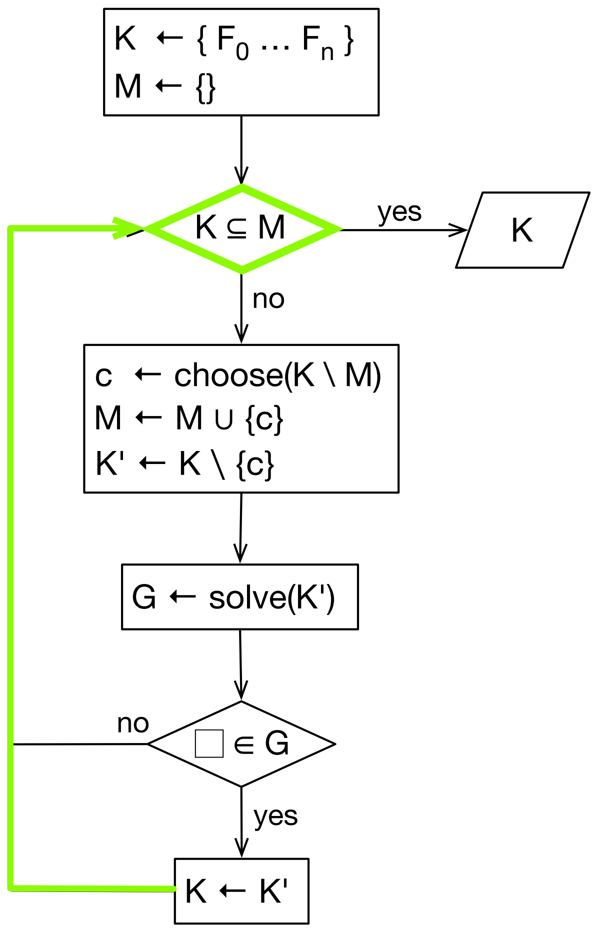


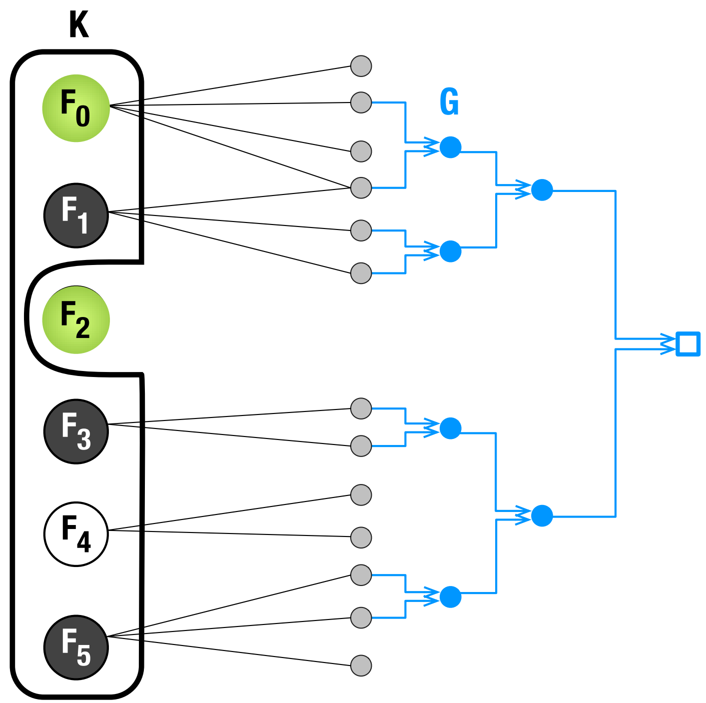
naive core extraction (NCE)



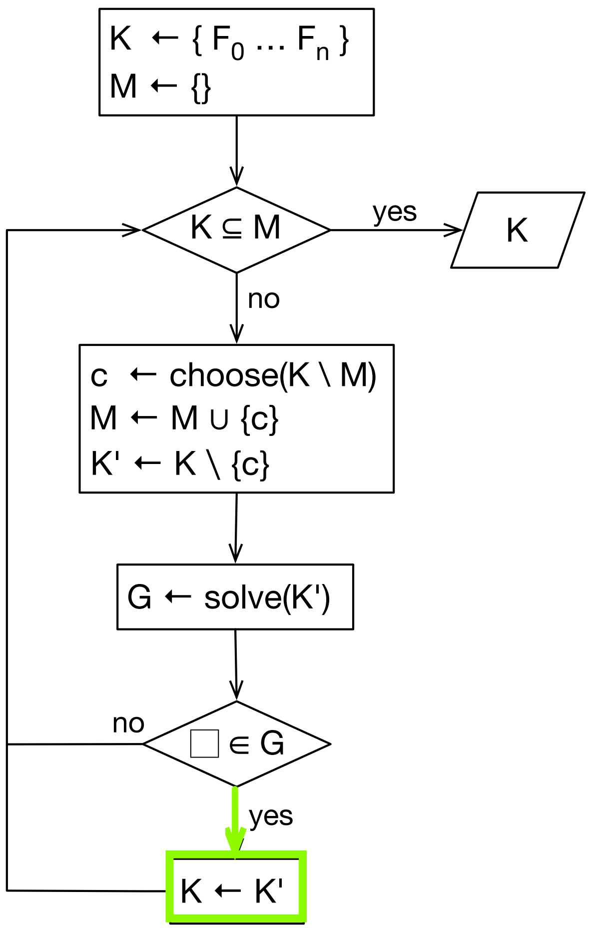


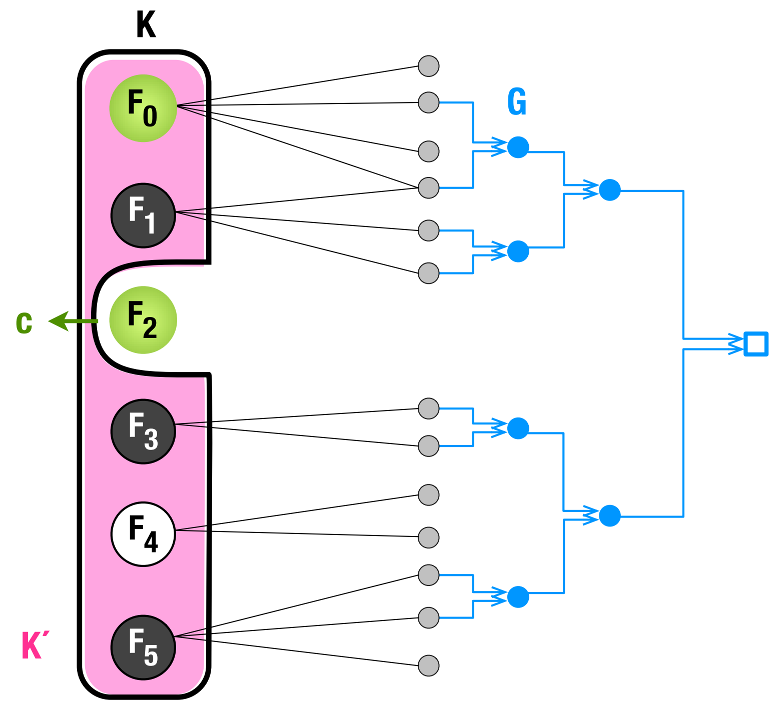
naive core extraction (NCE)



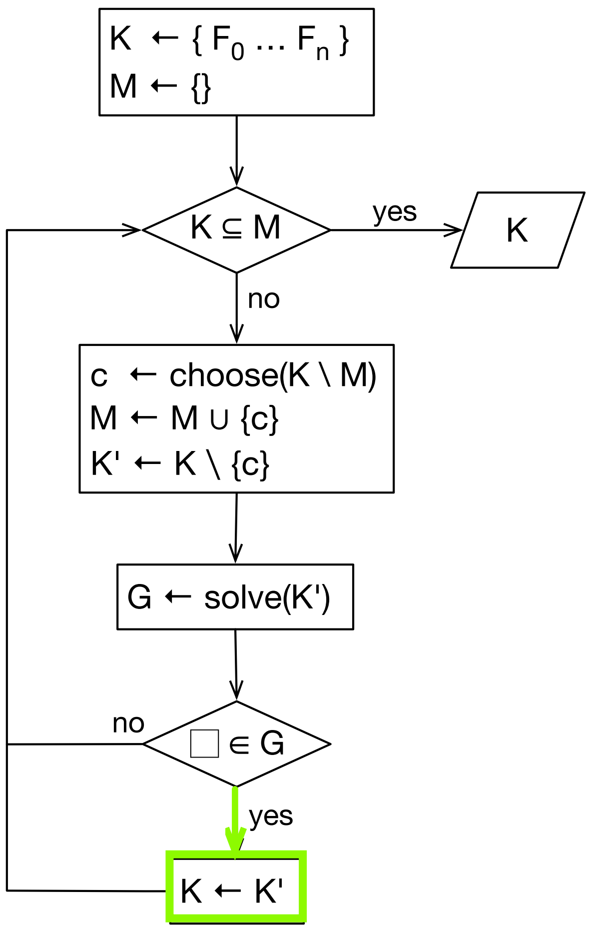


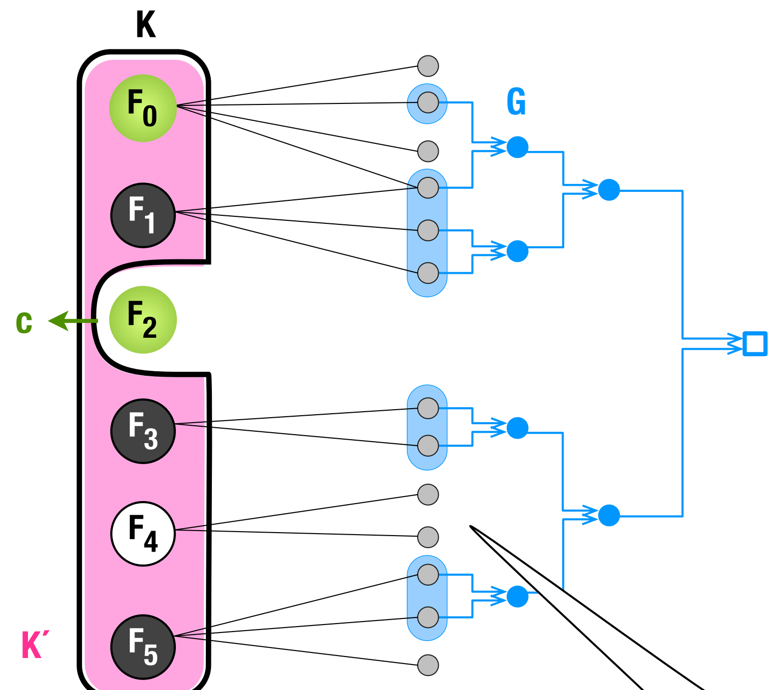
from naive to simple core





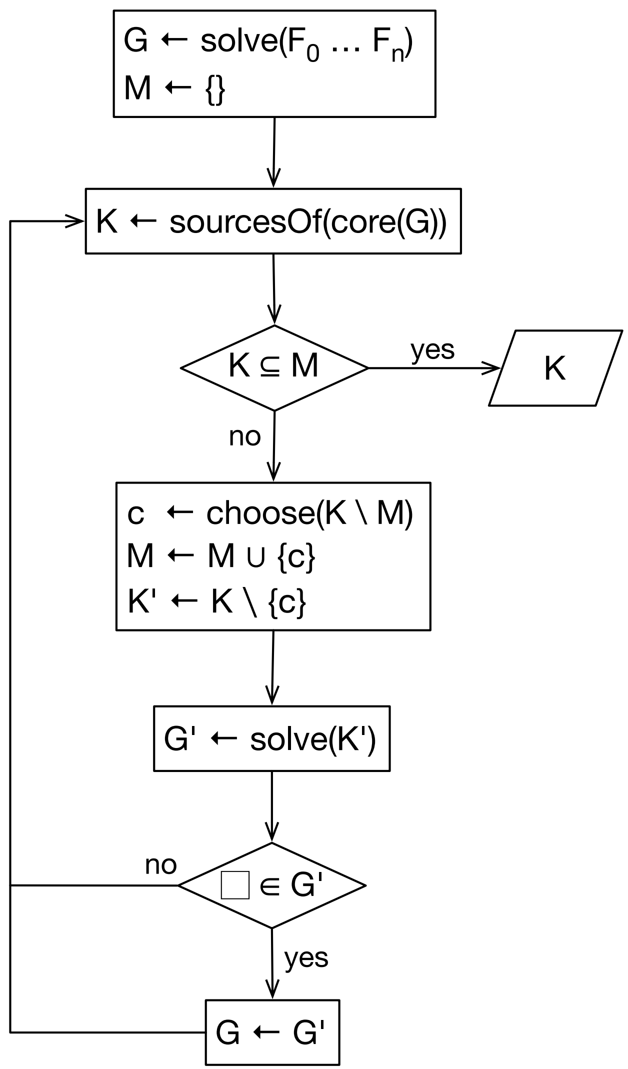
from naive to simple core



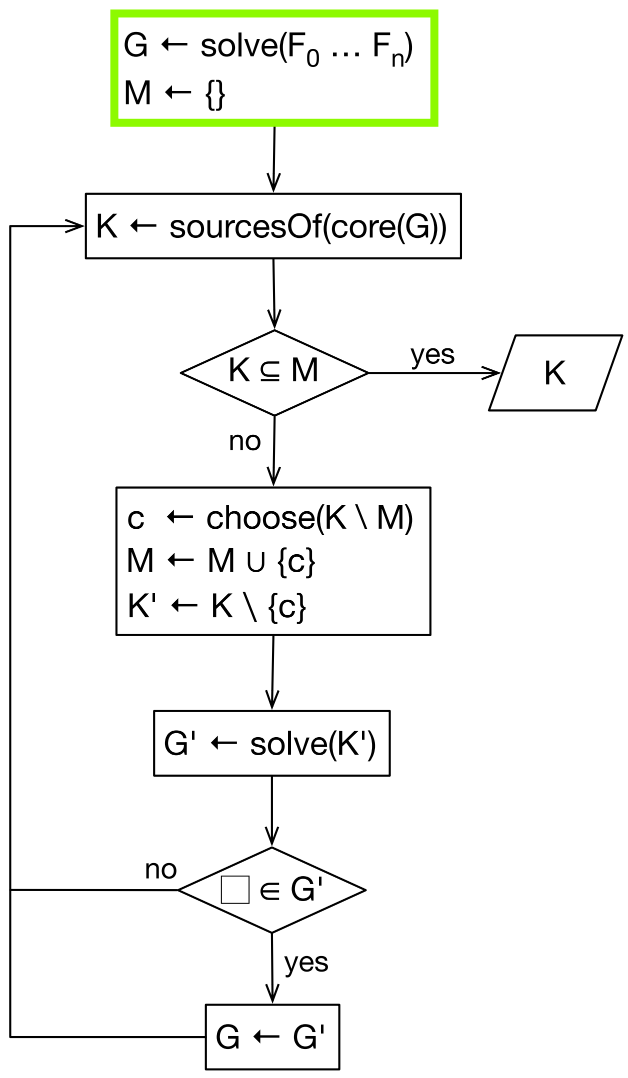


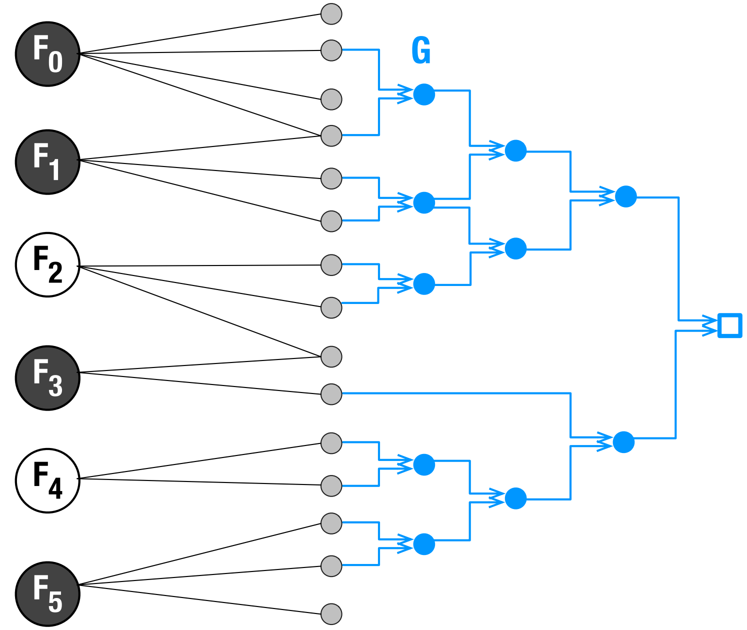
F4 not needed to prove the unsatisfiability of $K' = F_0 \wedge F_1 \wedge F_3 \wedge F_4 \wedge F_5$

simple core extraction (SCE)

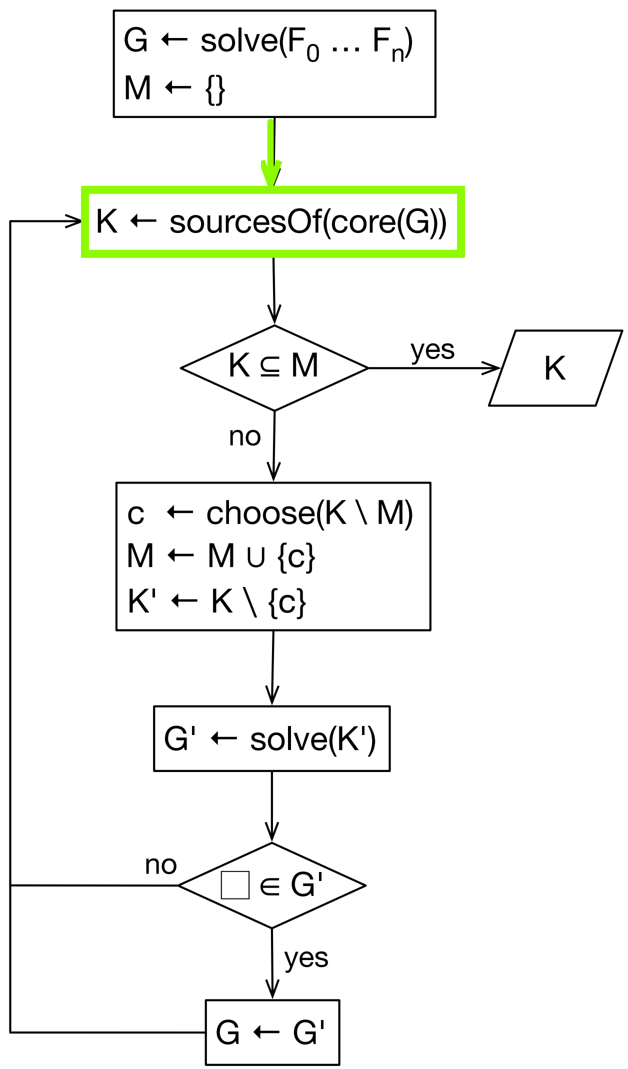


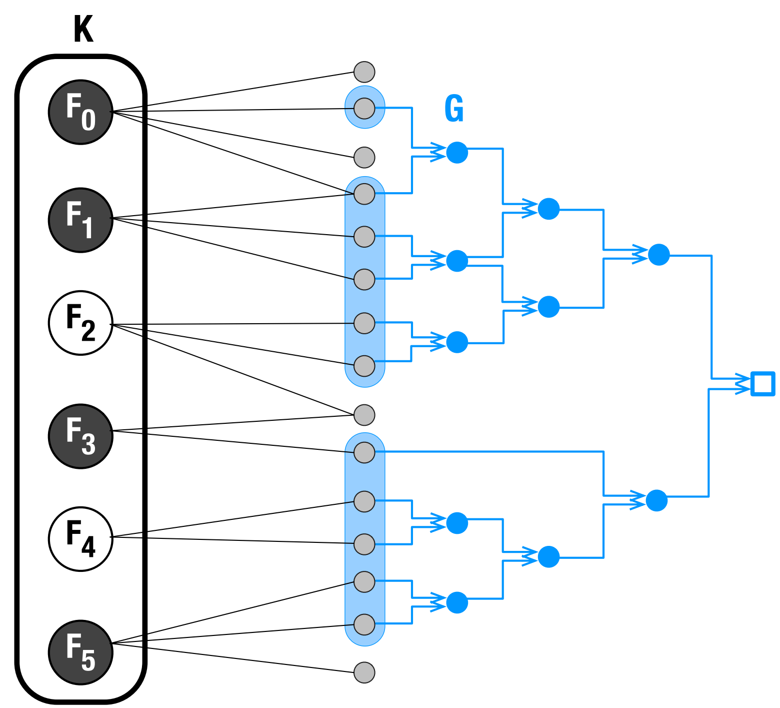
simple core extraction (SCE)



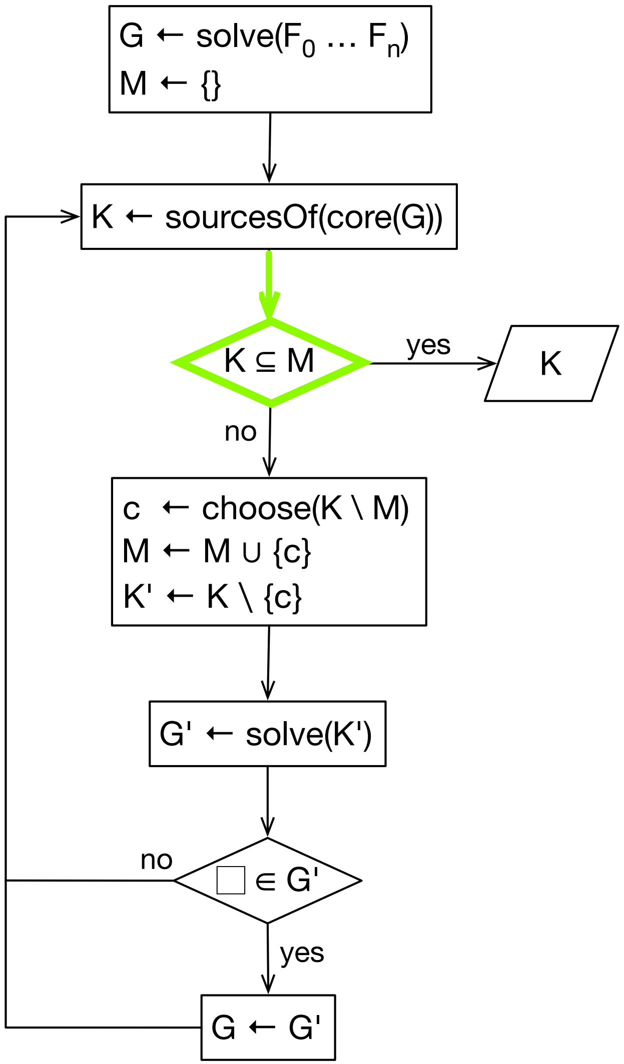


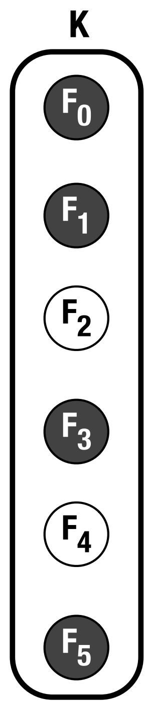
simple core extraction (SCE)

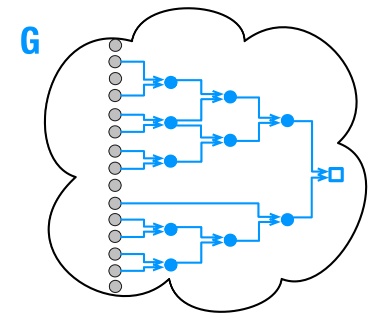




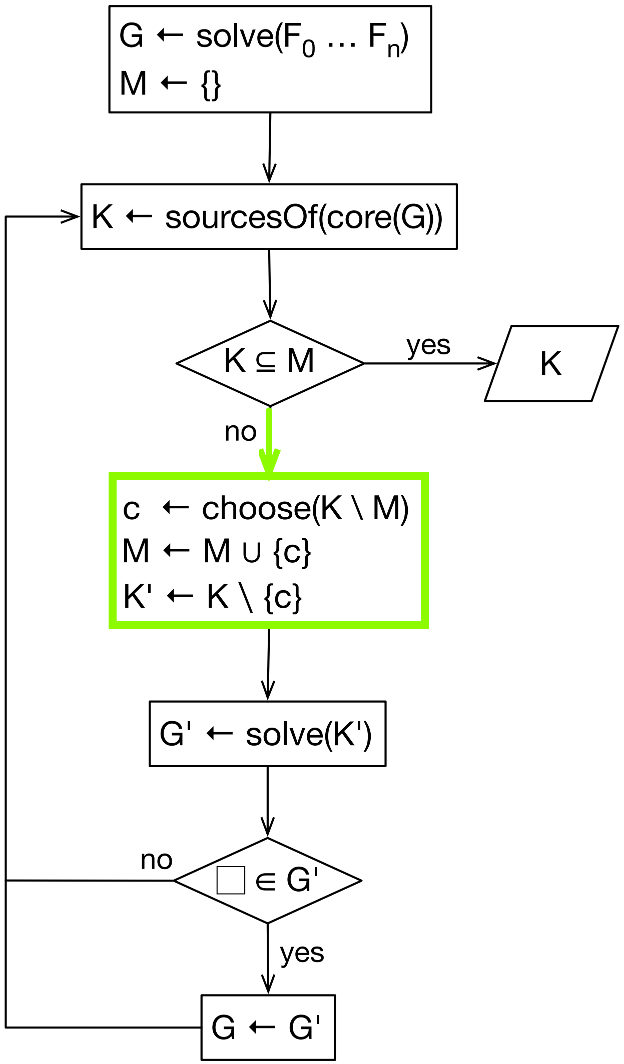
simple core extraction (SCE)

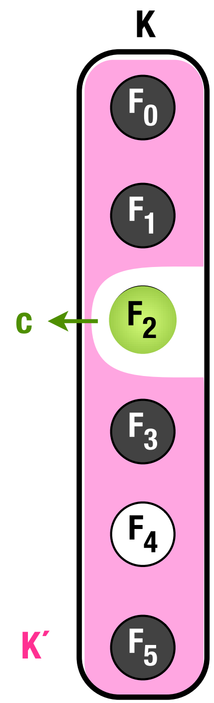


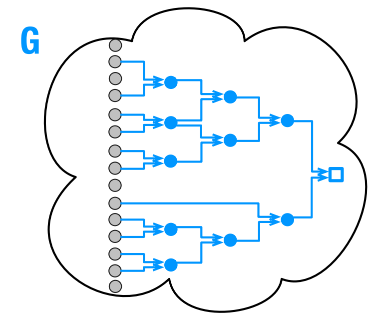




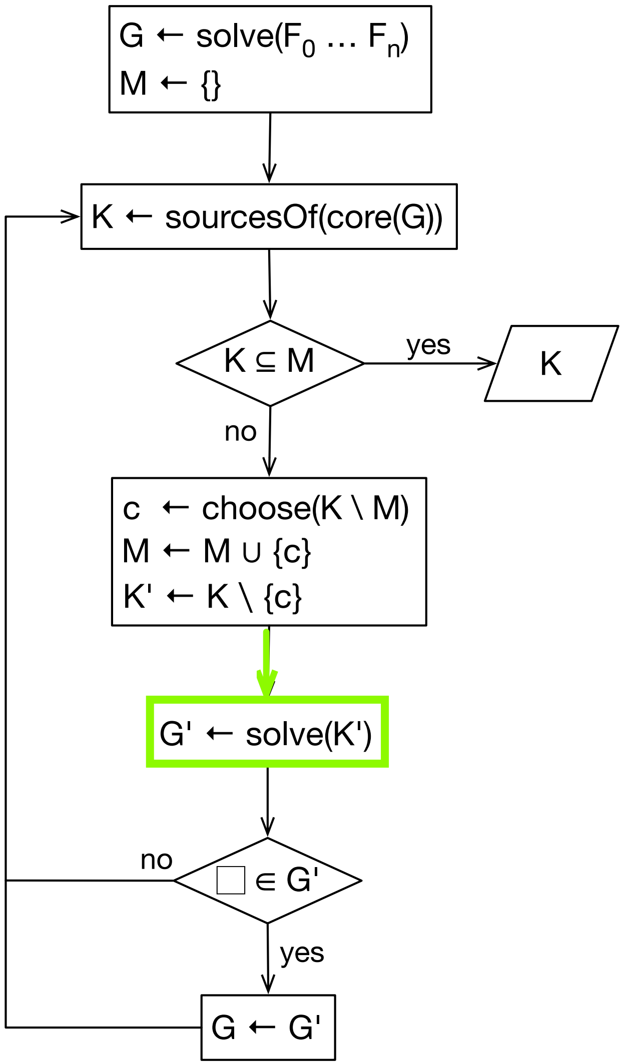
simple core extraction (SCE)

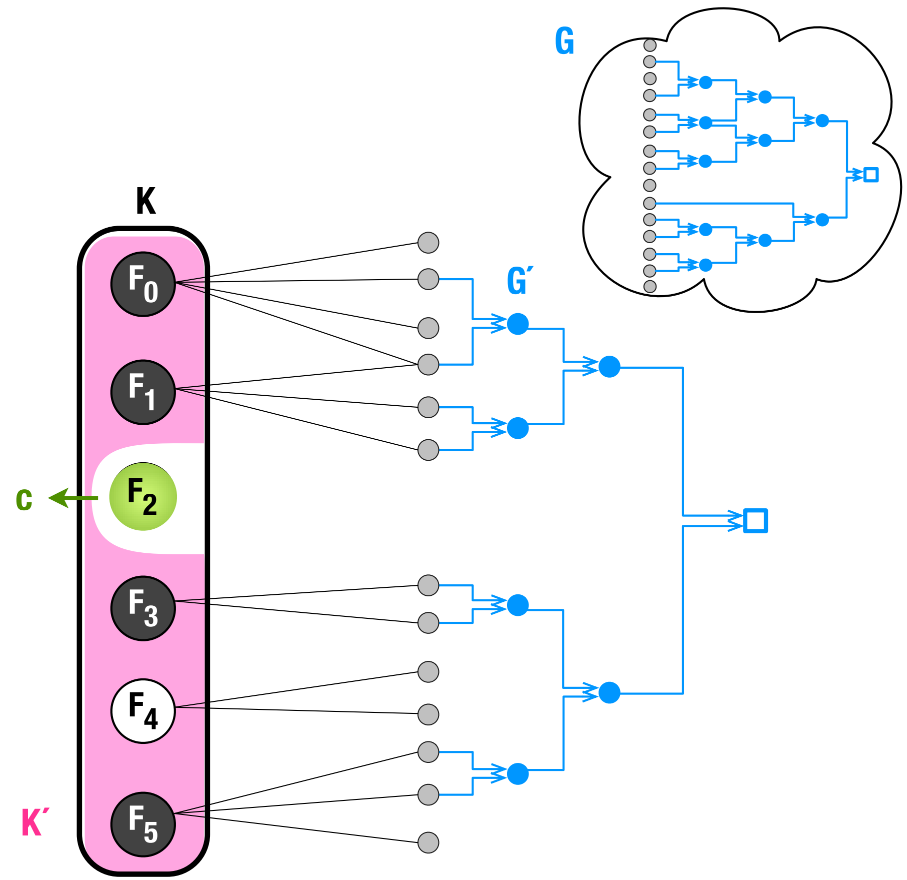




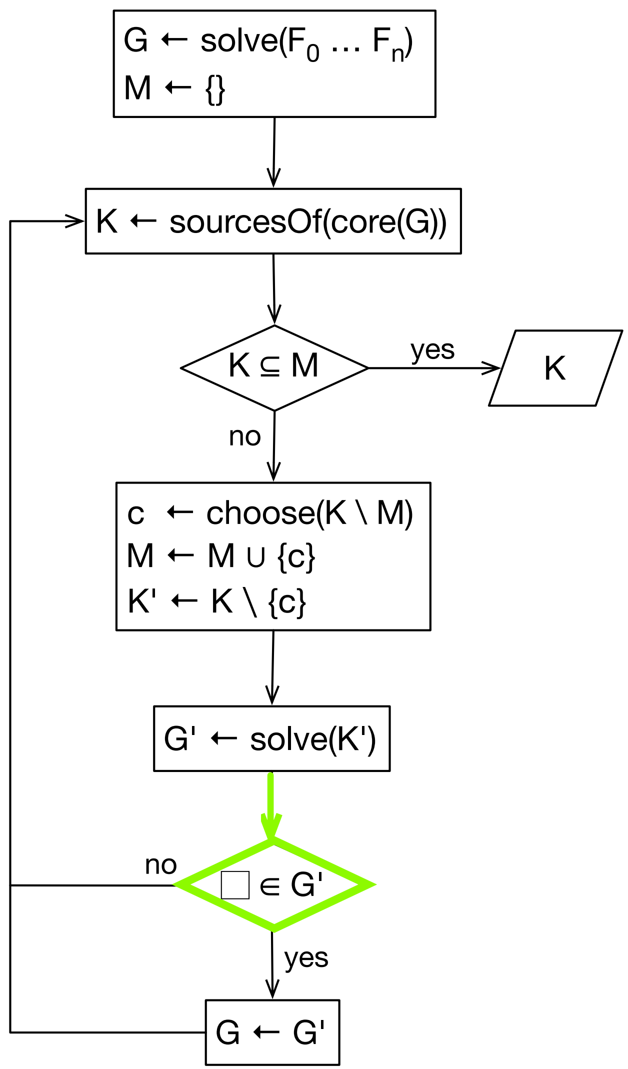


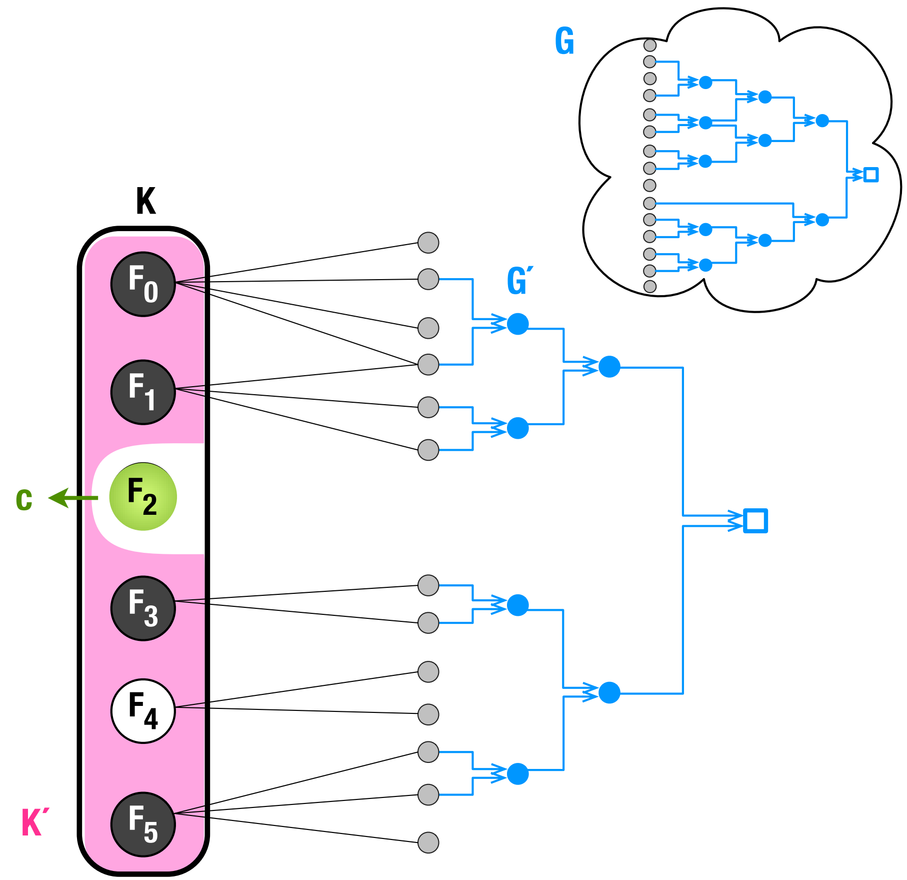
simple core extraction (SCE)



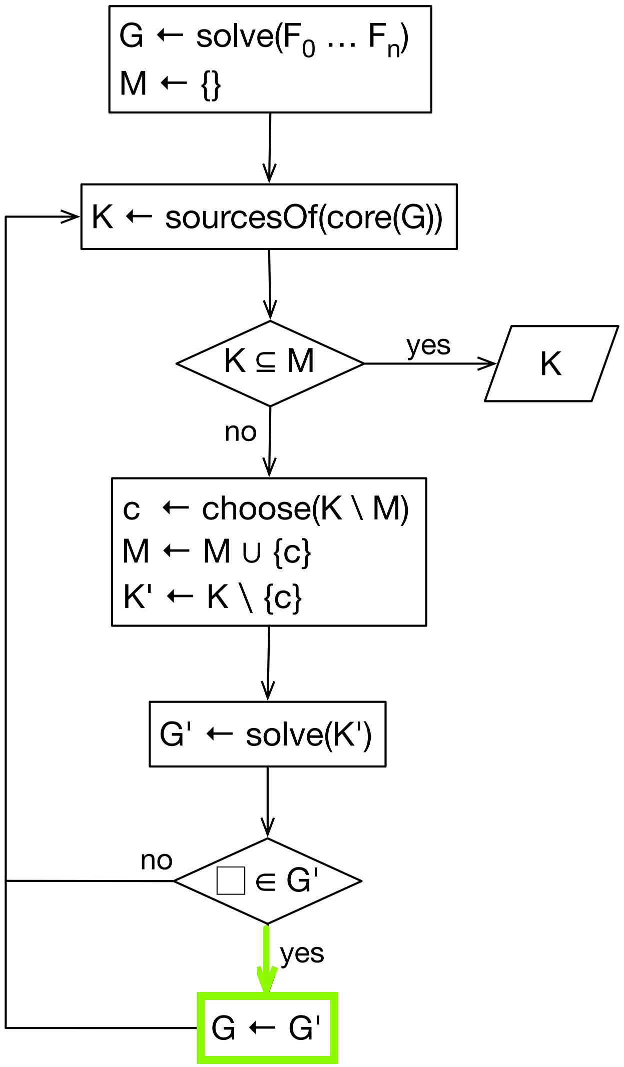


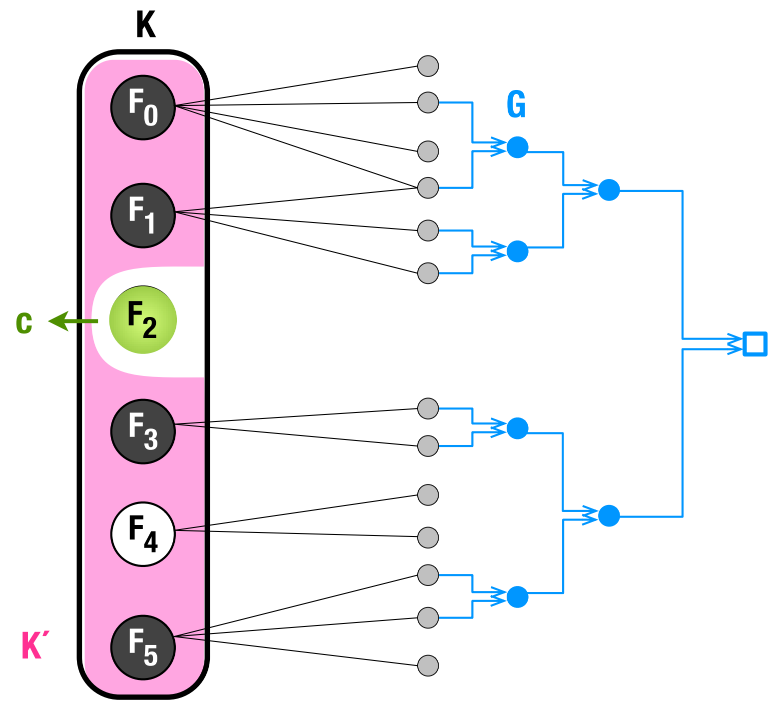
simple core extraction (SCE)



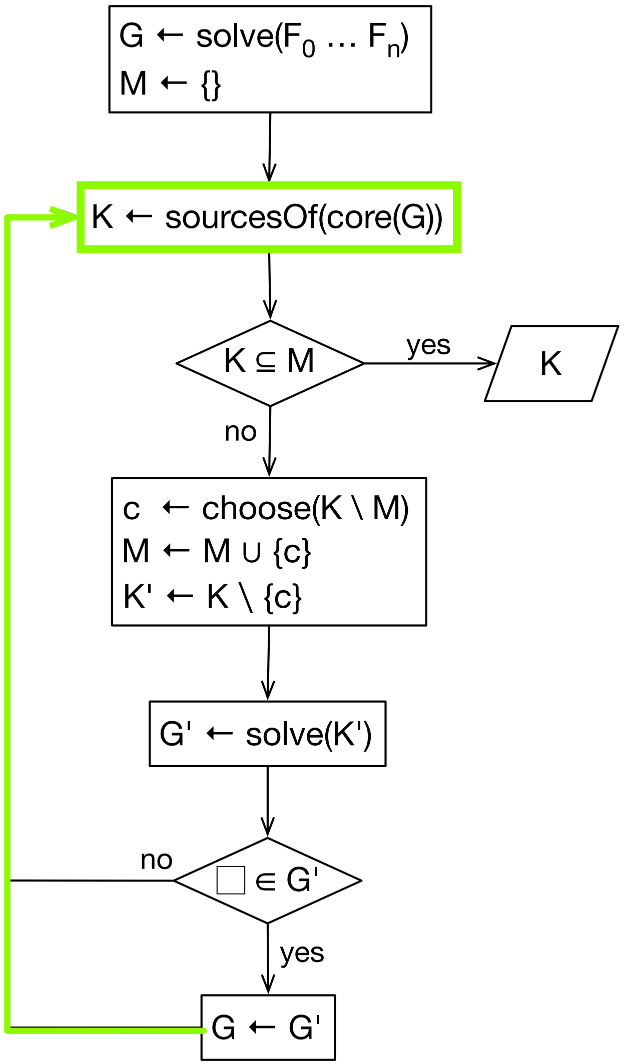


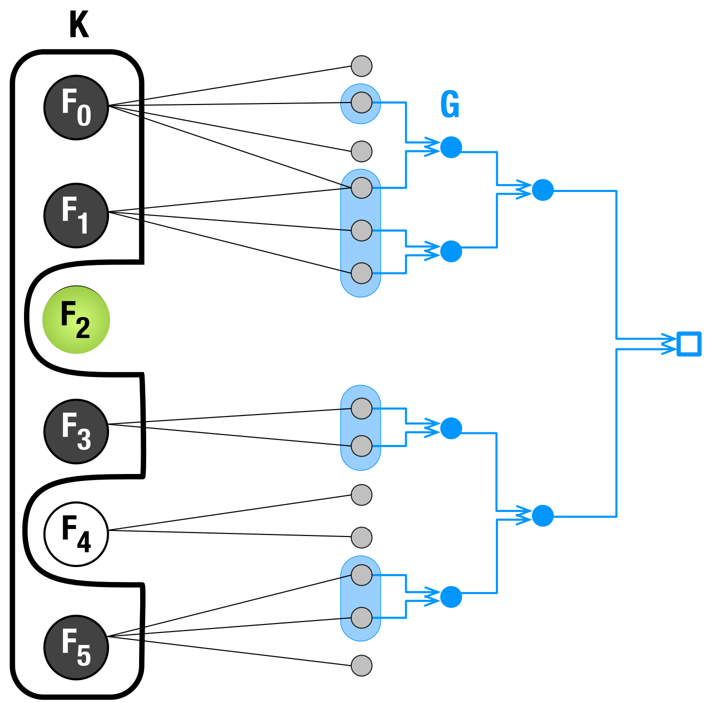
simple core extraction (SCE)



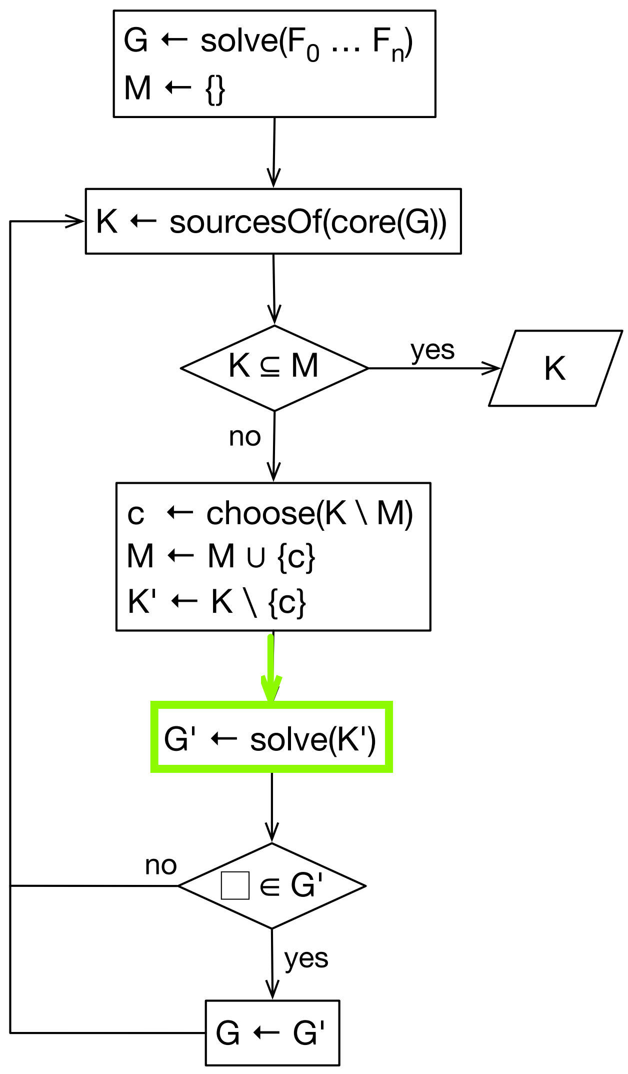


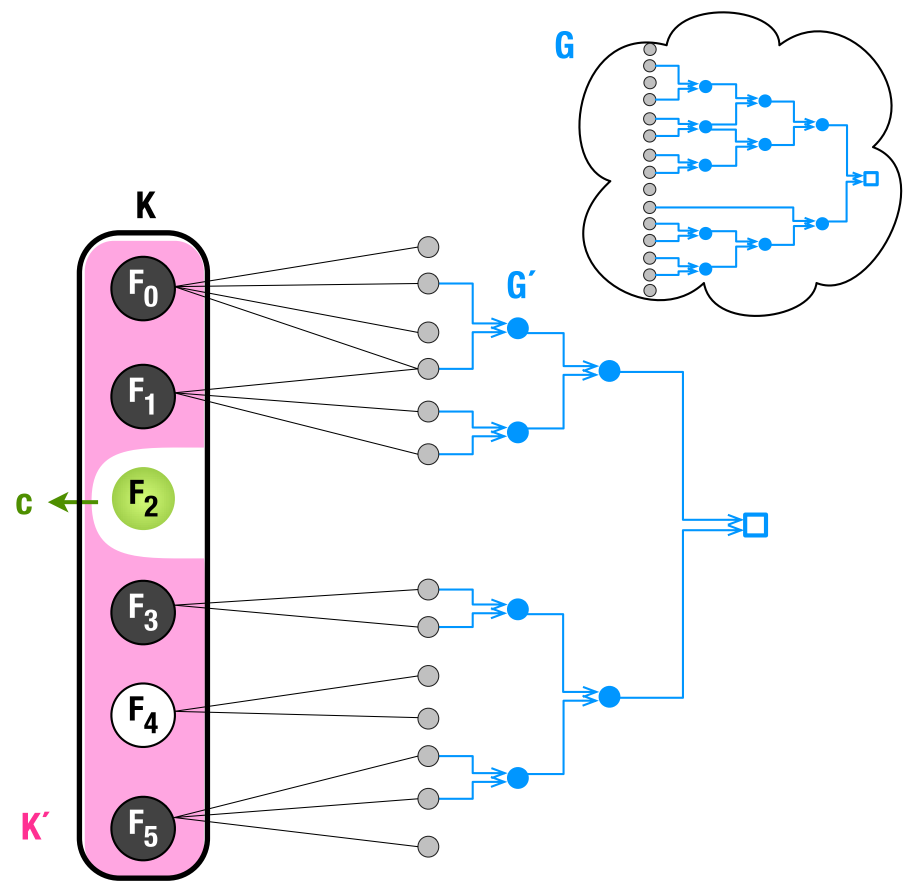
simple core extraction (SCE)



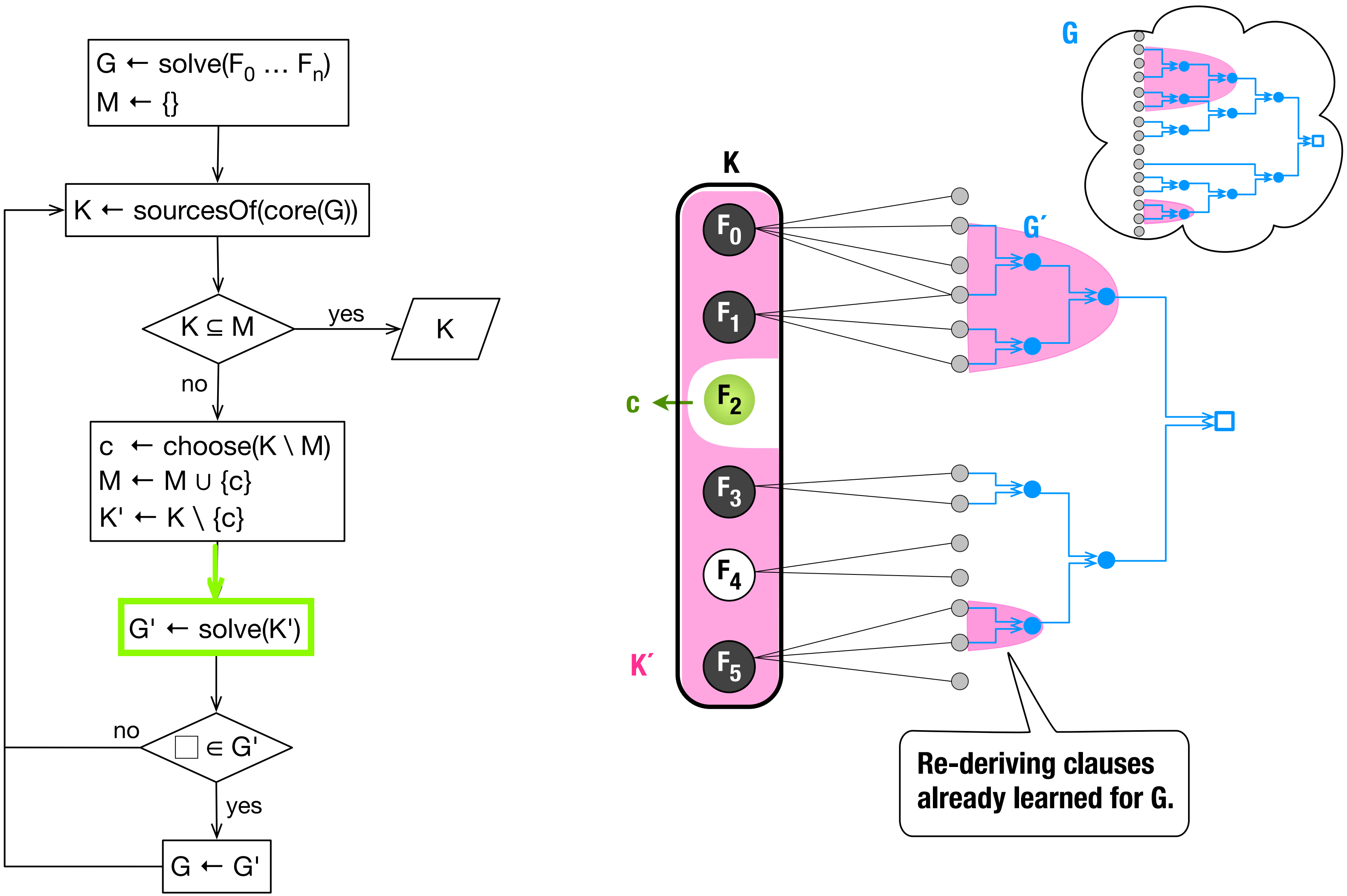


from simple to recycling core

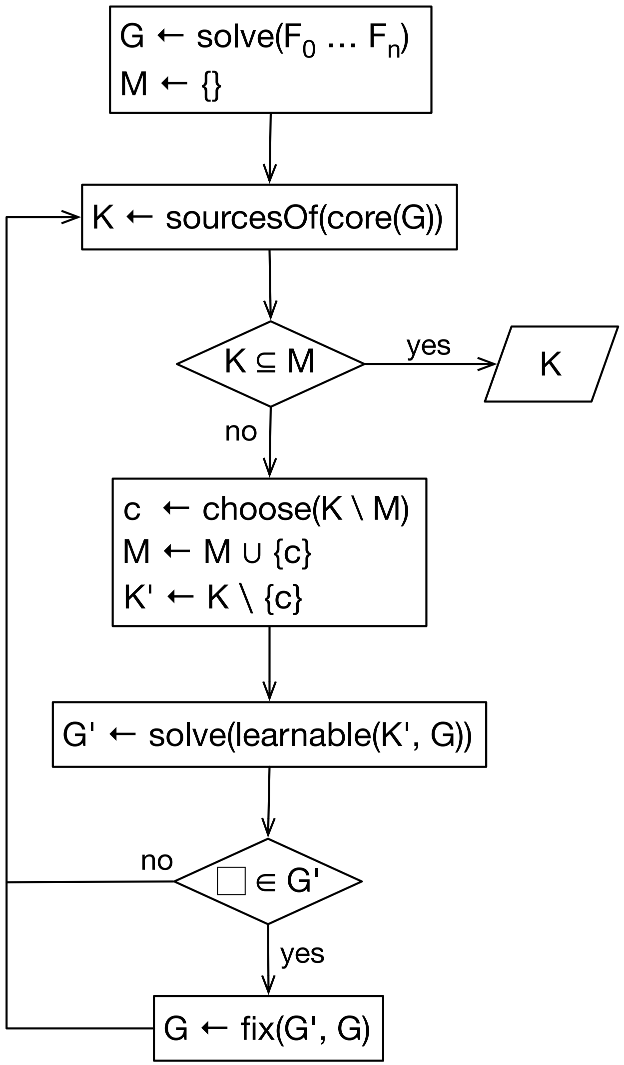


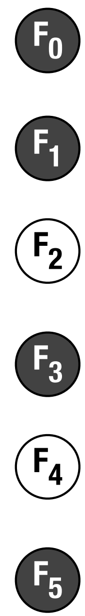


from simple to recycling core

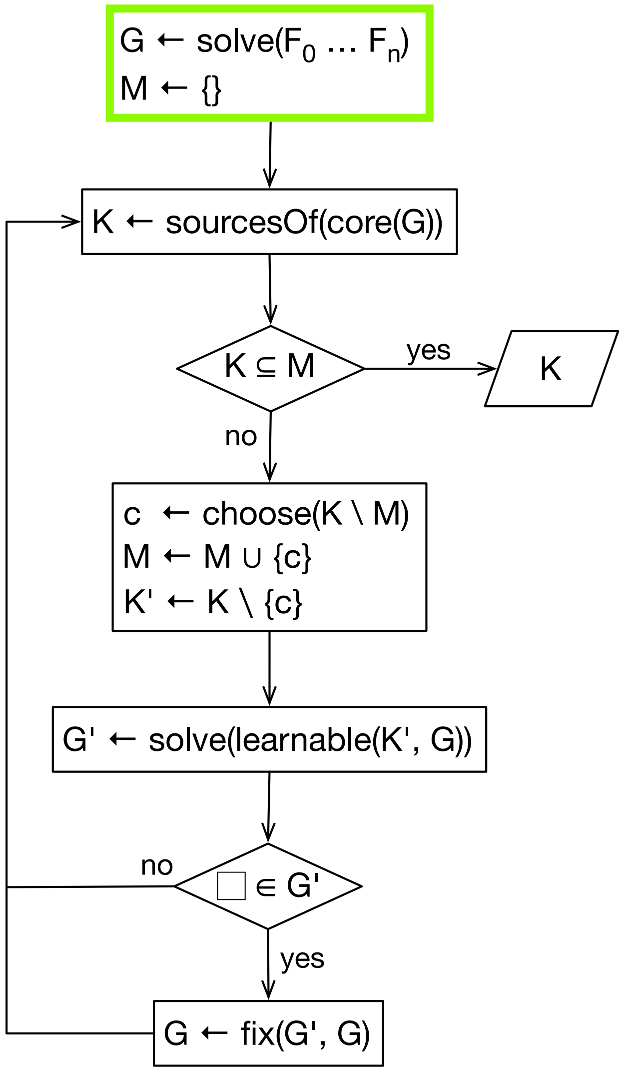


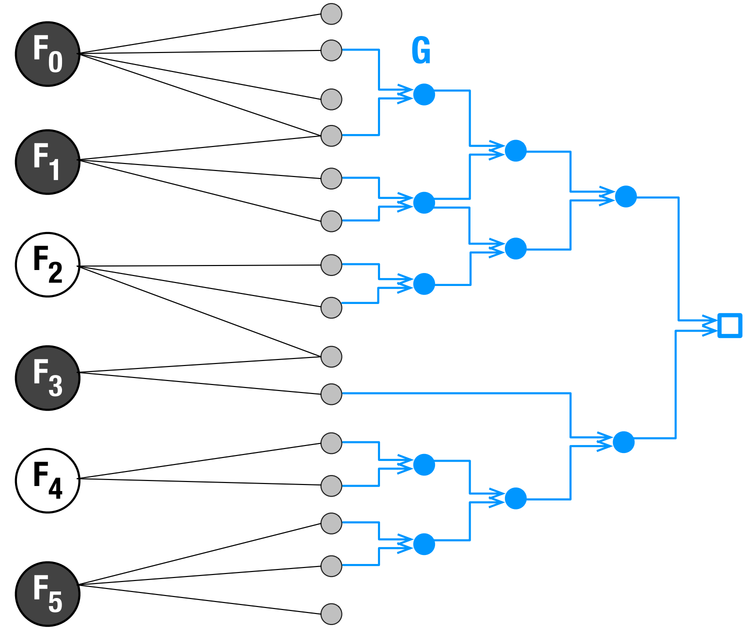
recycling core extraction (RCE)



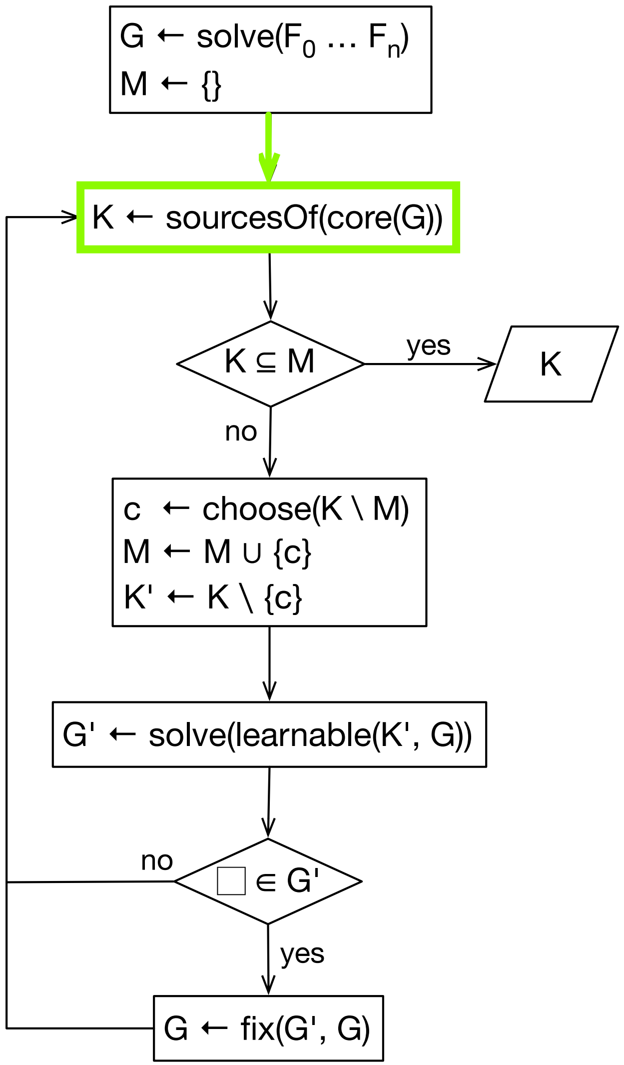


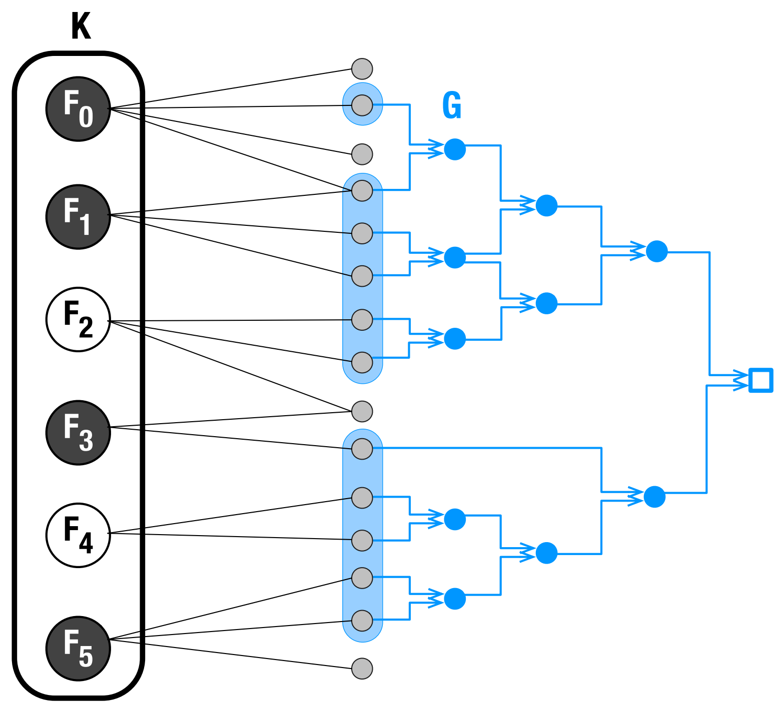
recycling core extraction (RCE)



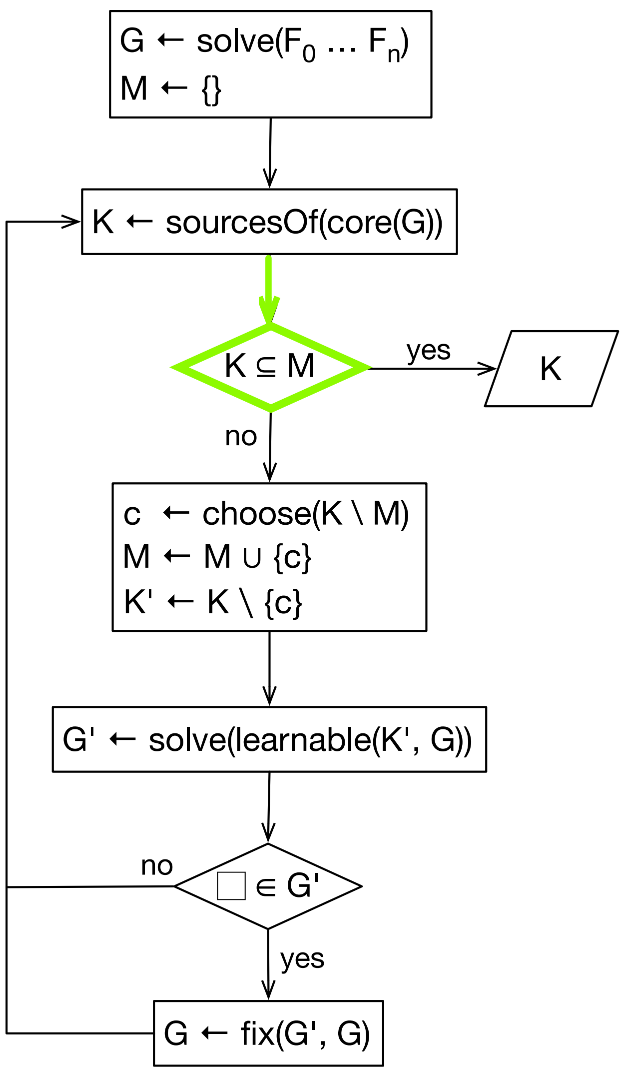


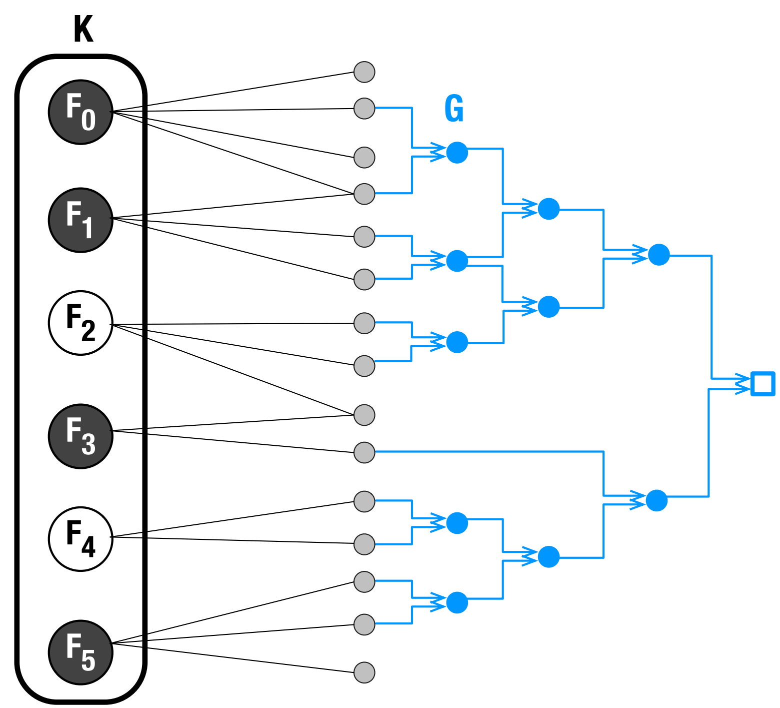
recycling core extraction (RCE)



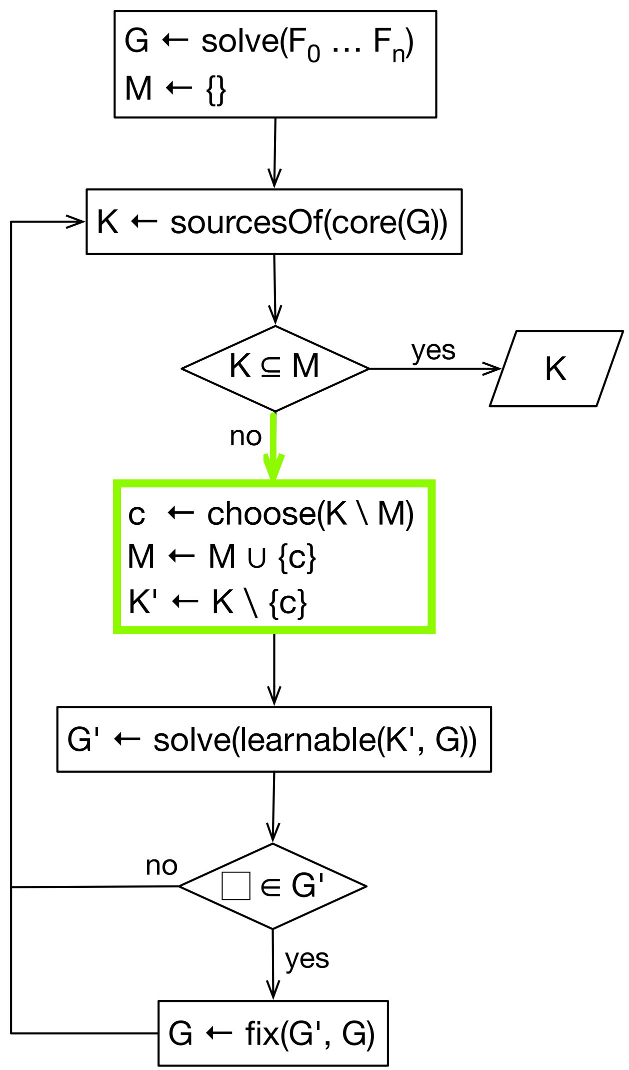


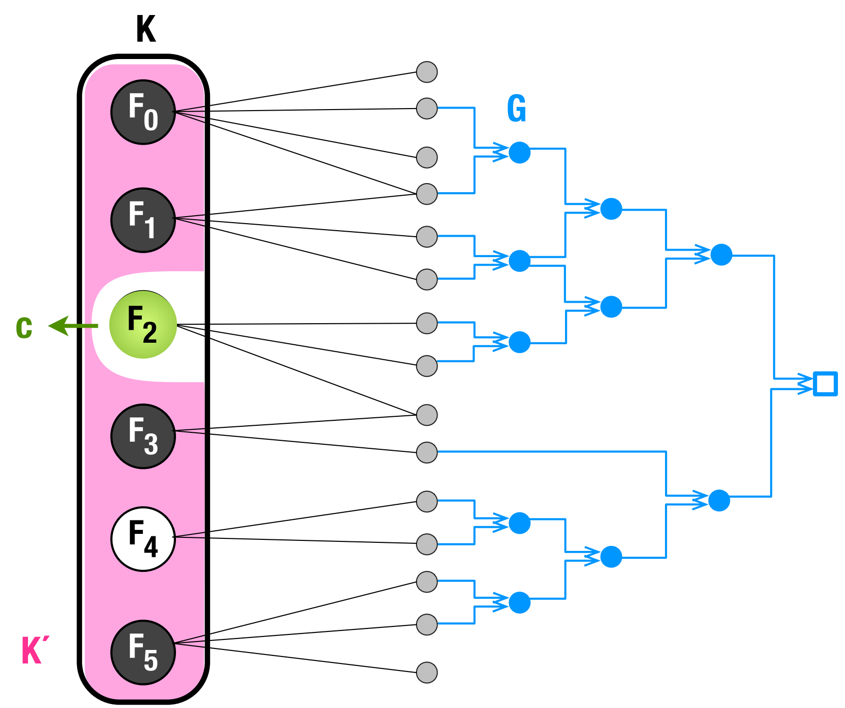
recycling core extraction (RCE)



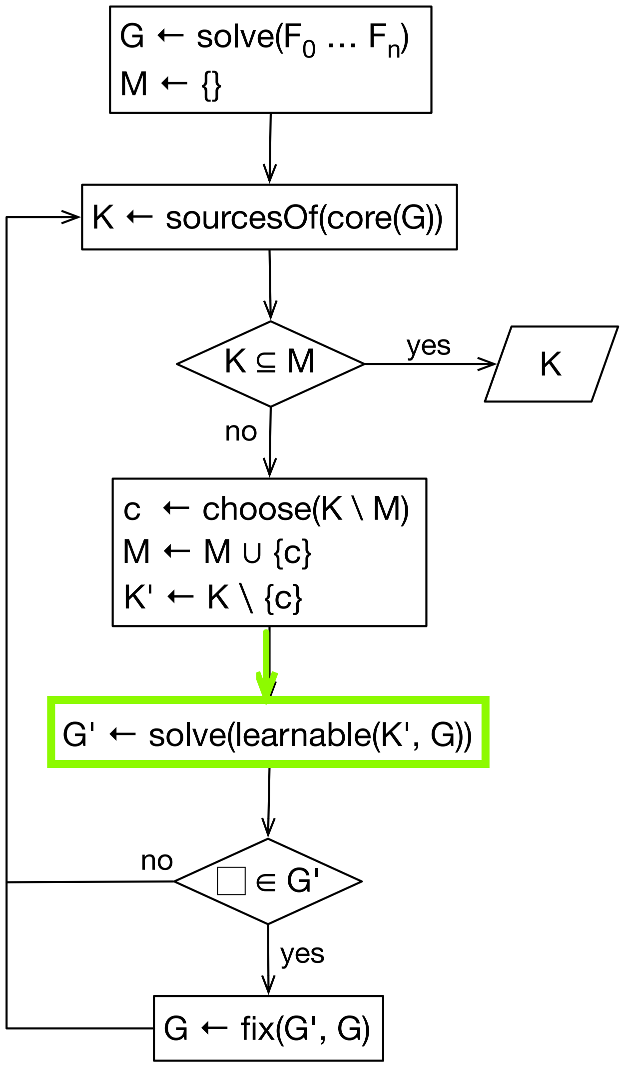


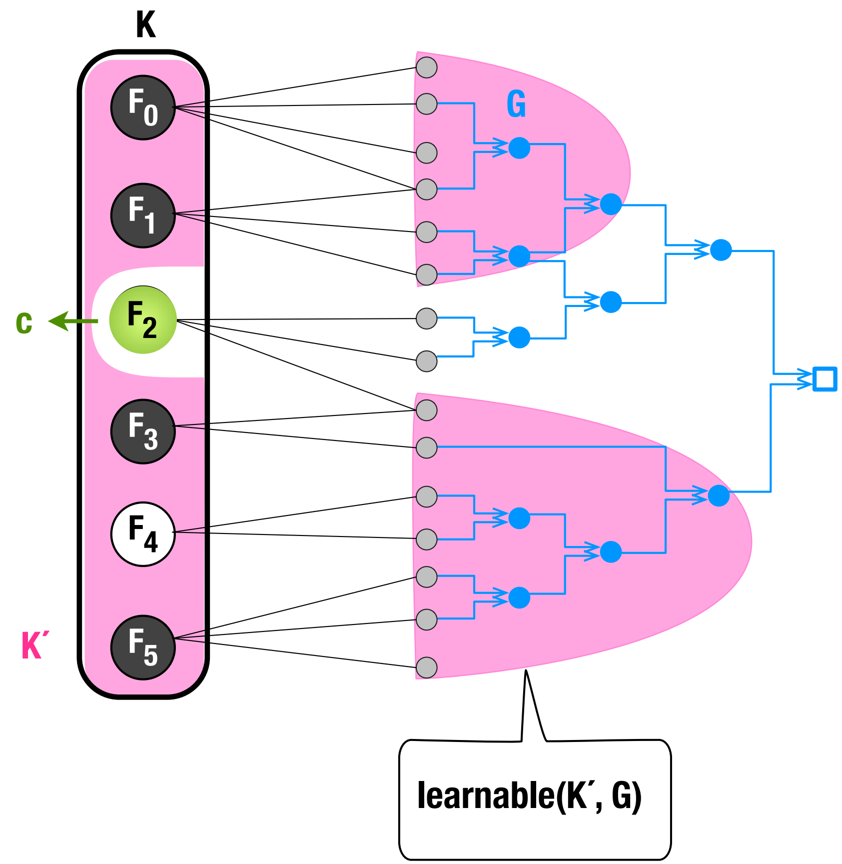
recycling core extraction (RCE)



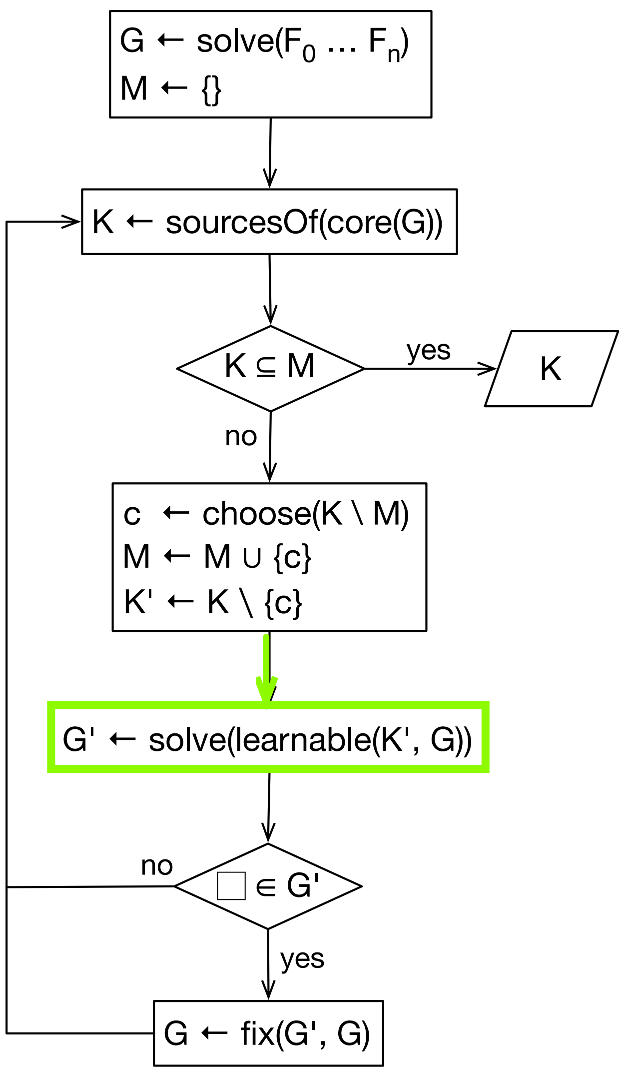


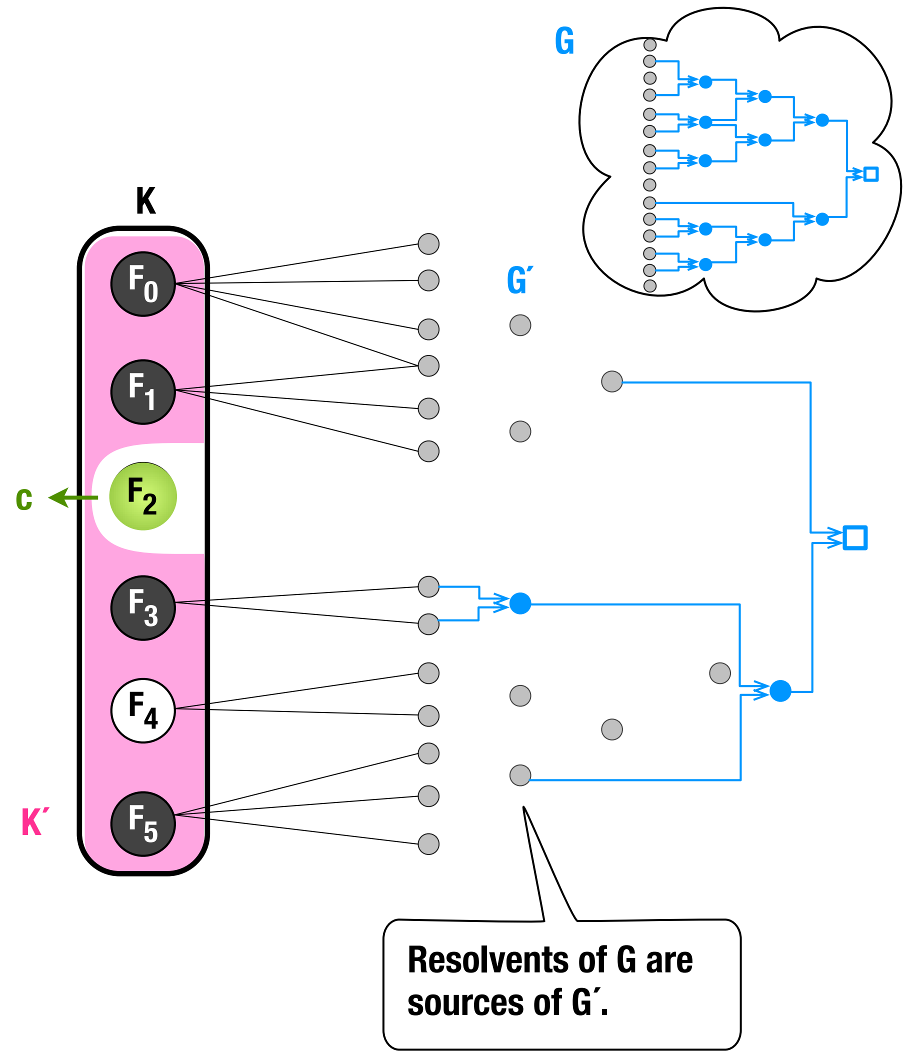
recycling core extraction (RCE)



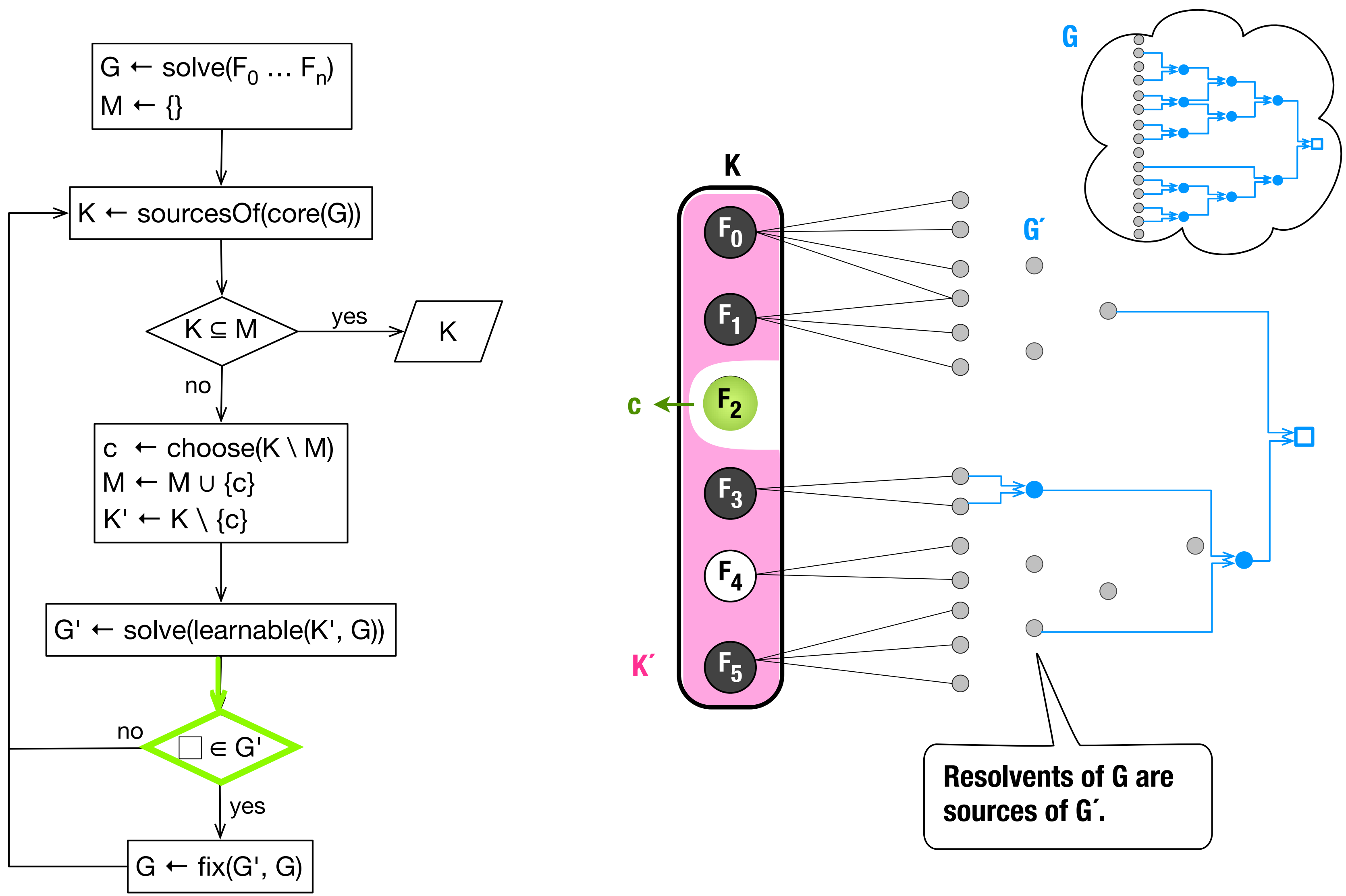


recycling core extraction (RCE)

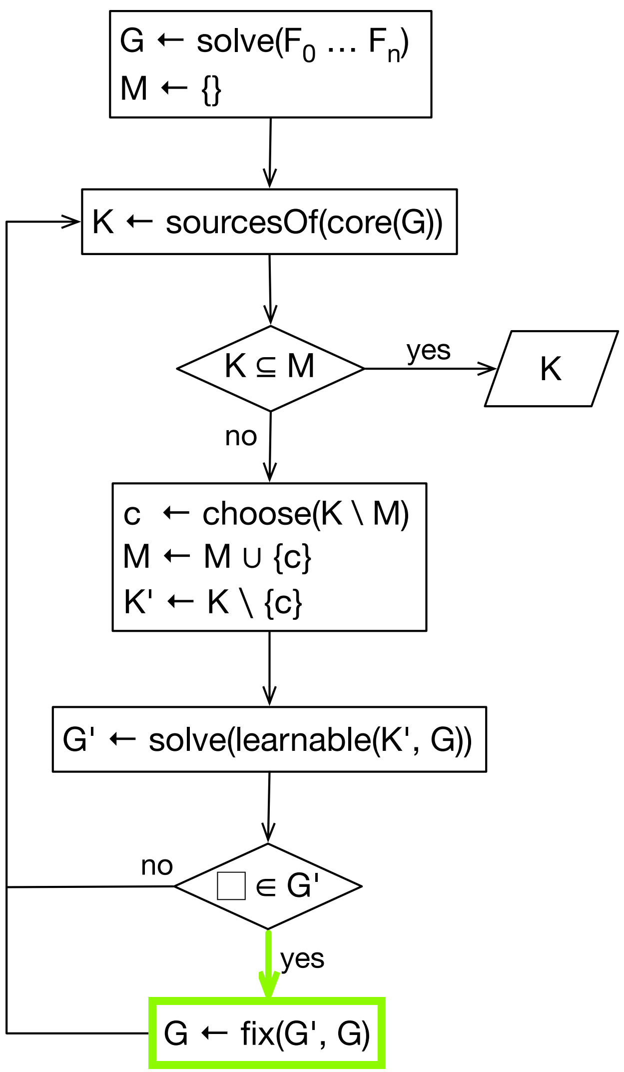


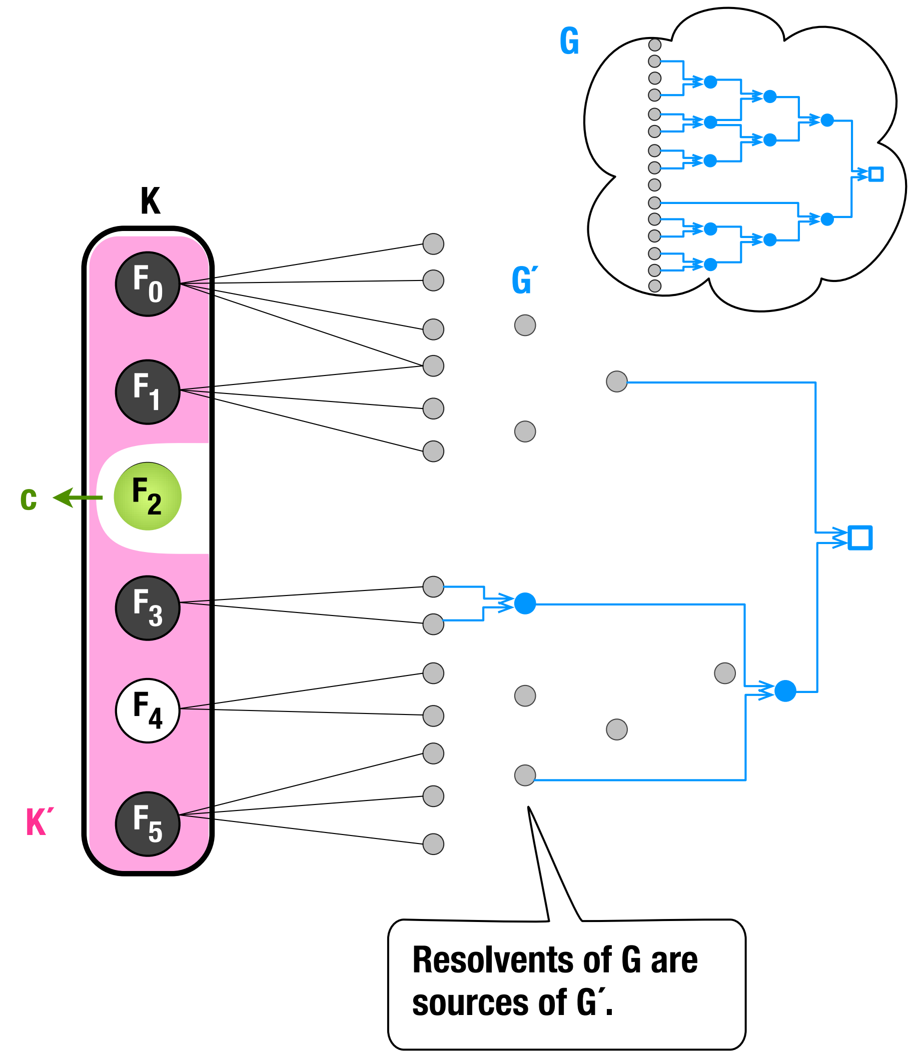


recycling core extraction (RCE)

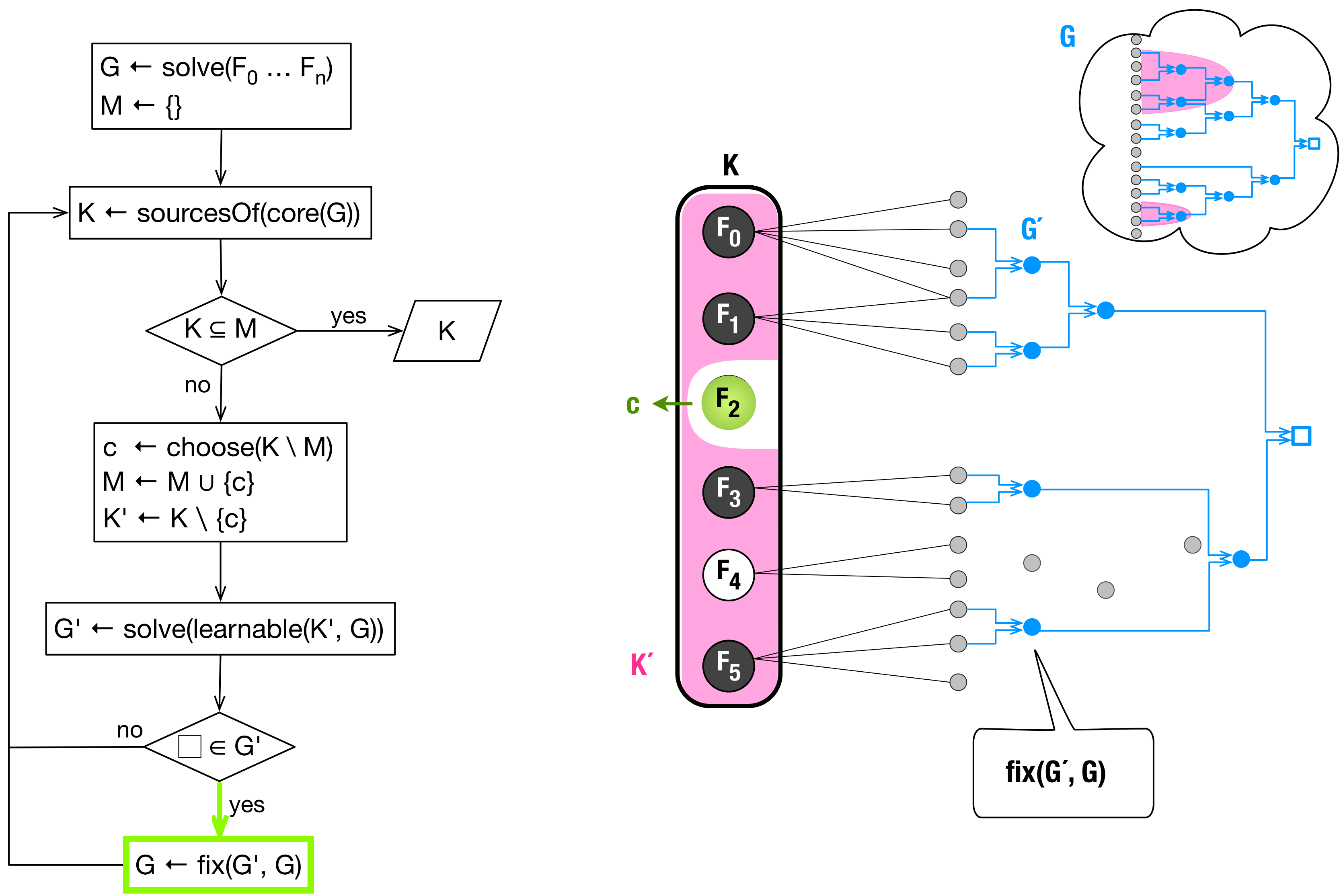


recycling core extraction (RCE)

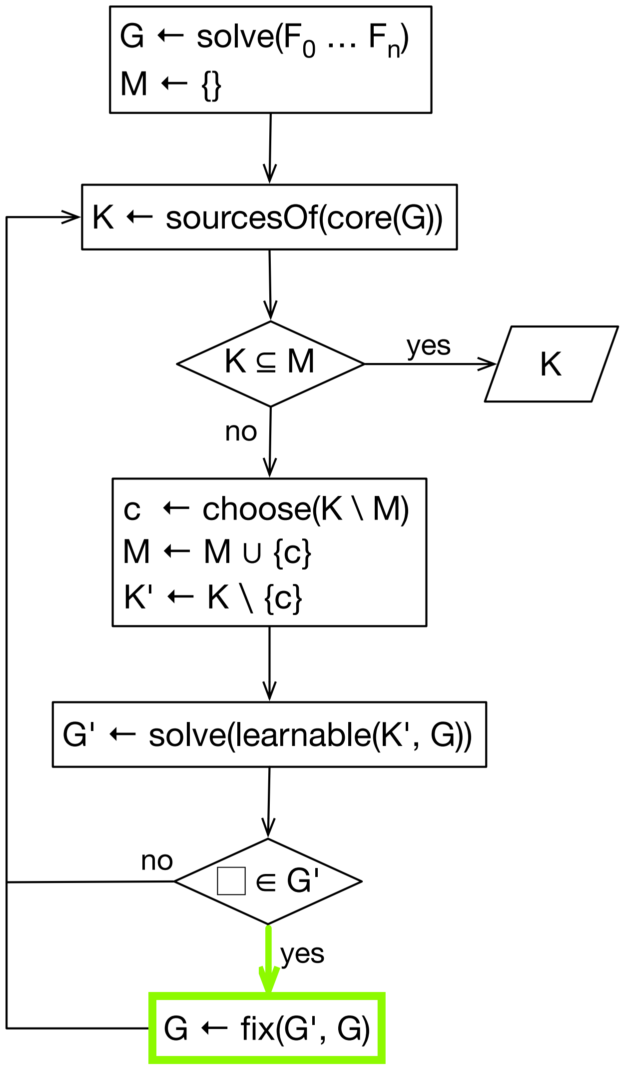


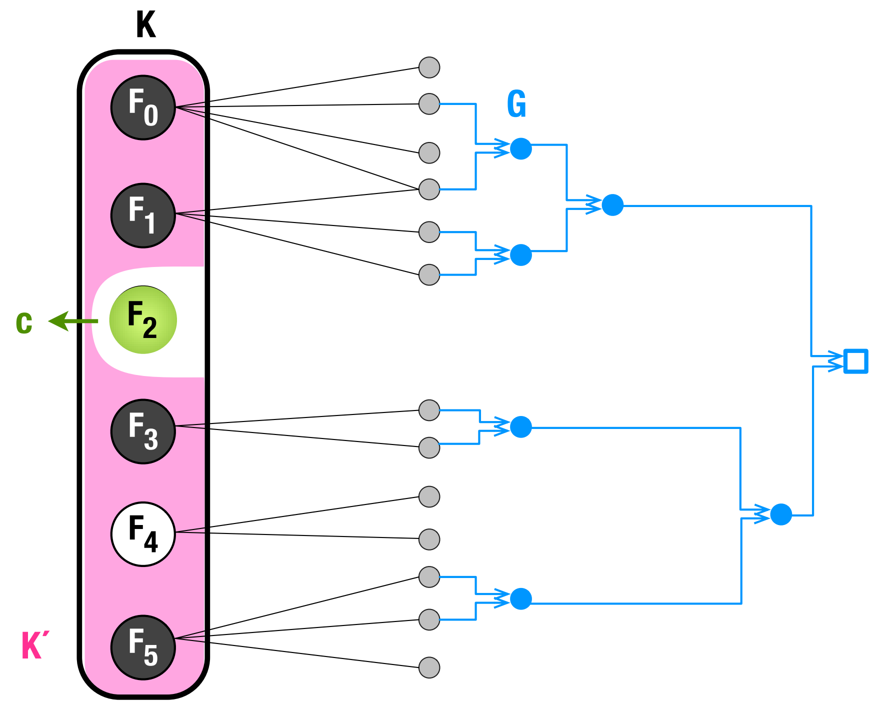


recycling core extraction (RCE)

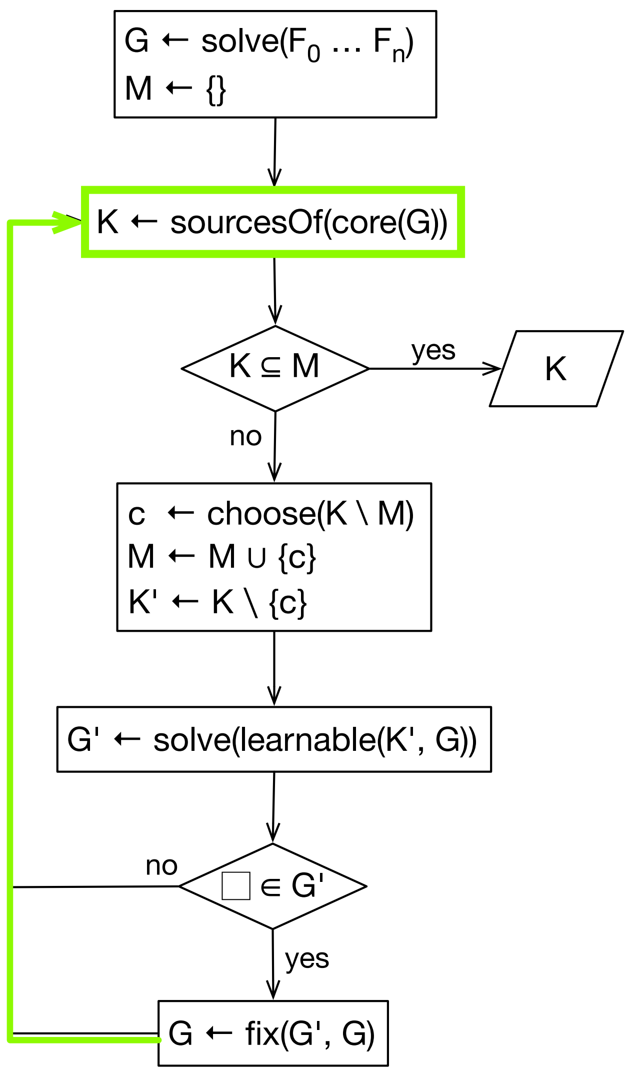


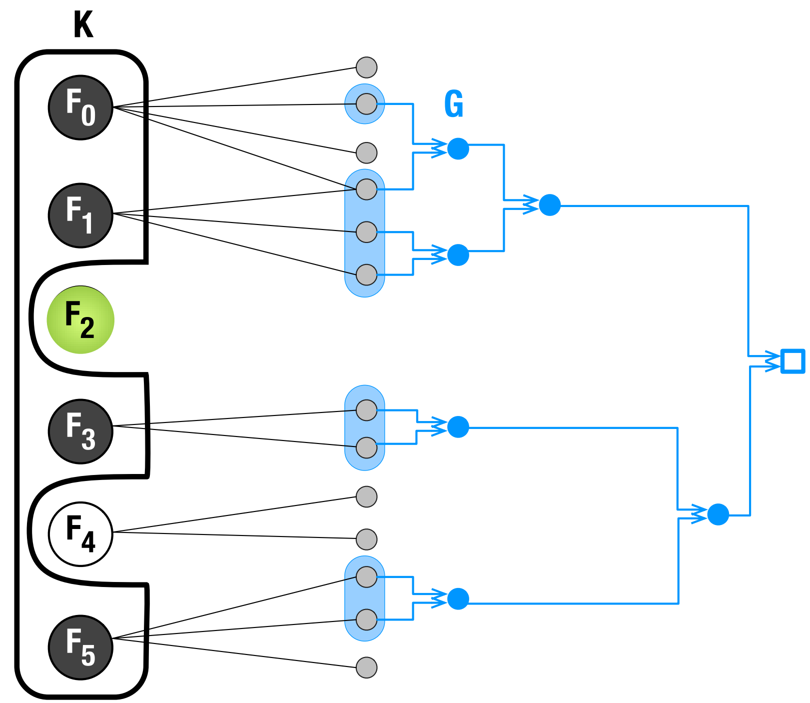
recycling core extraction (RCE)



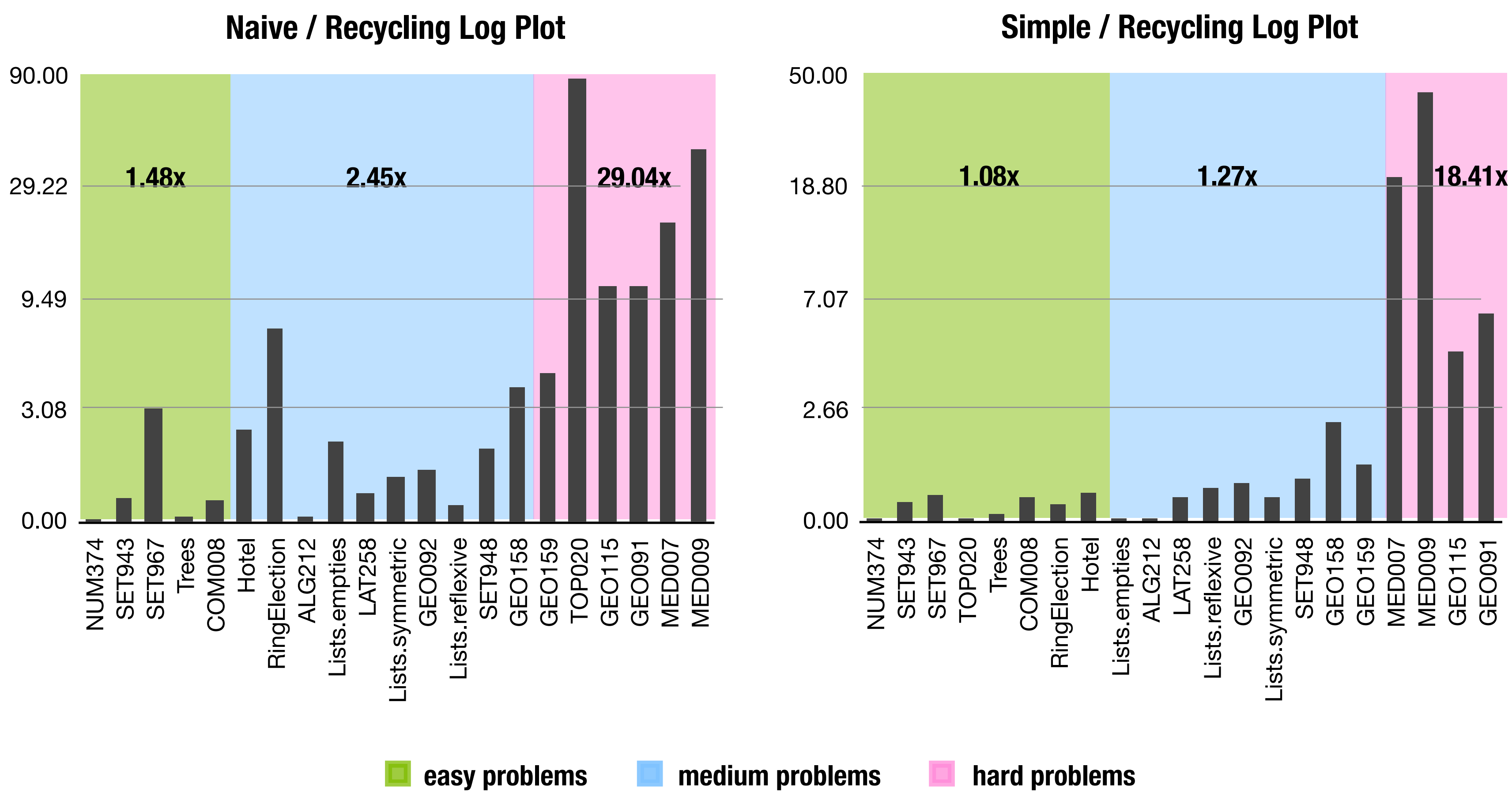


recycling core extraction (RCE)

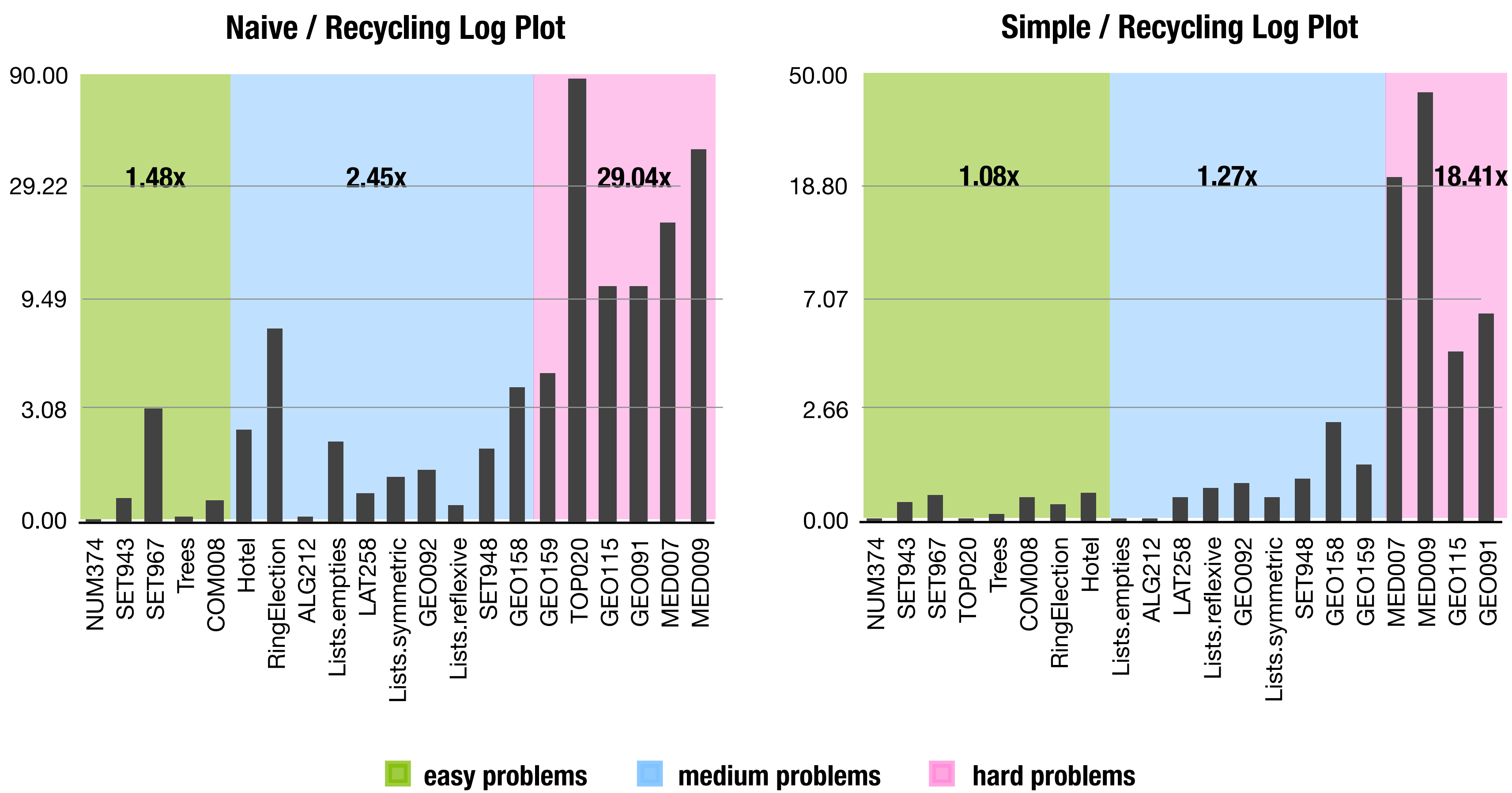




experimental results



experimental results



difficulty = #clauses to remove * time to solve

examples, revisited

ex1: overconstraint

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
   outcome =
      { n: Node, v: Value |
         let sn = { x: Node | all y: Node | x.id < y.id } |
            v = sn.vote
      }
   }

assert consensus {
   one v: Value | all n: Node | n.outcome = v
   some n: Node | n.outcome = n.vote
   }

check consensus for 3
```

```
Executing "Check consensus for 3"
  Solver=minisat(jni) Bitwidth=4 MaxSeq=3 SkolemDepth=2 Symmetry=20
  1153 vars. 72 primary vars. 3059 clauses. 29ms.
  No counterexample found. Assertion may be valid. 2ms.
```

core: property irrelevant

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node | x.id < y.id } |
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }

check consensus for 3
```

core: fixed, property relevant

```
sig Value {}
some sig Node { id: disj Int , vote, outcome: Value }

fact pickSmallest {
    outcome =
        { n: Node, v: Value |
            let sn = { x: Node | all y: Node - x | x.id < y.id }
                v = sn.vote
        }
    }

assert consensus {
    one v: Value | all n: Node | n.outcome = v
    some n: Node | n.outcome = n.vote
    }

check consensus for 3
```

ex2: weak check

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
  objects: set Object,
  links: Alias -> Object,
  contents: Dir -> Object
  }

abstract sig Action {
  pre, post: FileSystem
  }

sig DeleteAction extends Action {
  obj: Object
  } {
  post.objects = pre.objects - obj
  post.contents = pre.contents - obj->Object - Object->obj
  post.links = pre.links - Alias->obj - obj->Alias
  }

check {
  all d: DeleteAction | d.obj not in d.post.objects
  }
```

```
Executing "Check check$1"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  1893 vars. 108 primary vars. 4058 clauses. 38ms.
  No counterexample found. Assertion may be valid. 27ms.
```

core: action mostly irrelevant

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
    objects: set Object,
    links: Alias -> Object,
    contents: Dir -> Object
    }

abstract sig Action {
    pre, post: FileSystem
    }

sig DeleteAction extends Action {
    obj: Object
    } {
    post.objects = pre.objects - obj
    post.contents = pre.contents - obj->Object - Object->obj
    post.links = pre.links - Alias->obj - obj->Alias
    }

check {
    all d: DeleteAction | d.obj not in d.post.objects
    }
```

core: stronger property covers

```
abstract sig Object {}
sig File, Dir, Alias extends Object {}

sig FileSystem {
    objects: set Object,
    links: Alias -> Object,
    contents: Dir -> Object
    }

abstract sig Action {
    pre, post: FileSystem
    }

sig DeleteAction extends Action {
    obj: Object
    } {
    post.objects = pre.objects - obj
    post.contents = pre.contents - obj->Object - Object->obj
    post.links = pre.links - Alias->obj - obj->Object
    }

pred inv (fs: FileSystem) {
    fs.(links+contents).Object + Object.(fs.(contents+links)) in fs.objects
    }

check {
    all d: DeleteAction | d.pre.inv implies d.post.inv
    }
```

ex3: scope too small

```
sig Time {
   members: set Member
   }
sig Member {
   friend: Member -> Time
   }
abstract sig Event {
   pre, post: Time,
   m: Member
   }
sig Join extends Event {} {
   post.members = pre.members + m
   nochange [friend]
   }
sig Befriend extends Event {f: Member} {
   m + f in pre.members
   friend.post = friend.pre + m->f
   nochange [members]
   }

fact init {
   no first.members
   no friend.first
   }

assert Symmetric {
   all t: Time | friend.t = ~(friend.t)
   }
check Symmetric for 2
```

```
Executing "Check Symmetric for 2"
  Solver=minisatprover(jni) Bitwidth=4 MaxSeq=2 Skolem
  513 vars. 44 primary vars. 933 clauses. 17ms.
  No counterexample found. Assertion may be valid. 6ms.
```

core: friend update irrelevant

```
sig Time {
    members: set Member
    }
sig Member {
    friend: Member -> Time
    }
abstract sig Event {
    pre, post: Time,
    m: Member
    }
sig Join extends Event {} {
    post.members = pre.members + m
    nochange [friend]
    }
sig Befriend extends Event {f: Member} {
    m + f in pre.members
    friend.post = friend.pre + m->f
    nochange [members]
    }

fact init {
    no first.members
    no friend.first
    }

assert Symmetric {
    all t: Time | friend.t = ~(friend.t)
    }
check Symmetric for 2
```

core: fixed, now relevant

```
sig Time {
    members: set Member
    }
sig Member {
    friend: Member -> Time
    }
abstract sig Event {
    pre, post: Time,
    m: Member
    }
sig Join extends Event {} {
    post.members = pre.members + m
    nochange [friend]
    }
sig Befriend extends Event {f: Member} {
    m + f in pre.members
    friend.post = friend.pre + m->f + f->m
    nochange [members]
    }

fact init {
    no first.members
    no friend.first
    }

assert Symmetric {
    all t: Time | friend.t = ~(friend.t)
    }
check Symmetric for 4
```

challenges

sometimes too slow

- in worst case, like solving k times when k top-level formulas

what does mincore mean for source?

- what's a formula?
- currently, only top-level conjuncts

  $P(x) \land Q(x) \lor R(x)$

- and some splitting of quantifiers

  $\forall x \mid P(x) \land Q(x) \Rightarrow \forall x \mid P(x) \land \forall x \mid Q(x)$

conclusions

Nelson: “sorry I can’t find any more bugs”

- even truer than we once thought
- problem not limited to bounded analysis

over-constraint detection based on unsat core

- a uniform solution for a range of problems
- other applications: eg, to configuration errors [Narain, Telcordia]

give it a try!

- try Alloy or add unsat core to your own tool with Kodkod
- download Alloy and Kodkod at http://alloy.mit.edu