VSR-NET Workshop - May 25th-26th, 2006
RAL Cosener's House
Abingdon, UK

Alloy / Mondex Case Study : Refinement Checks with Model Finding

Tahina Ramananandro
École Normale Supérieure
Paris, France

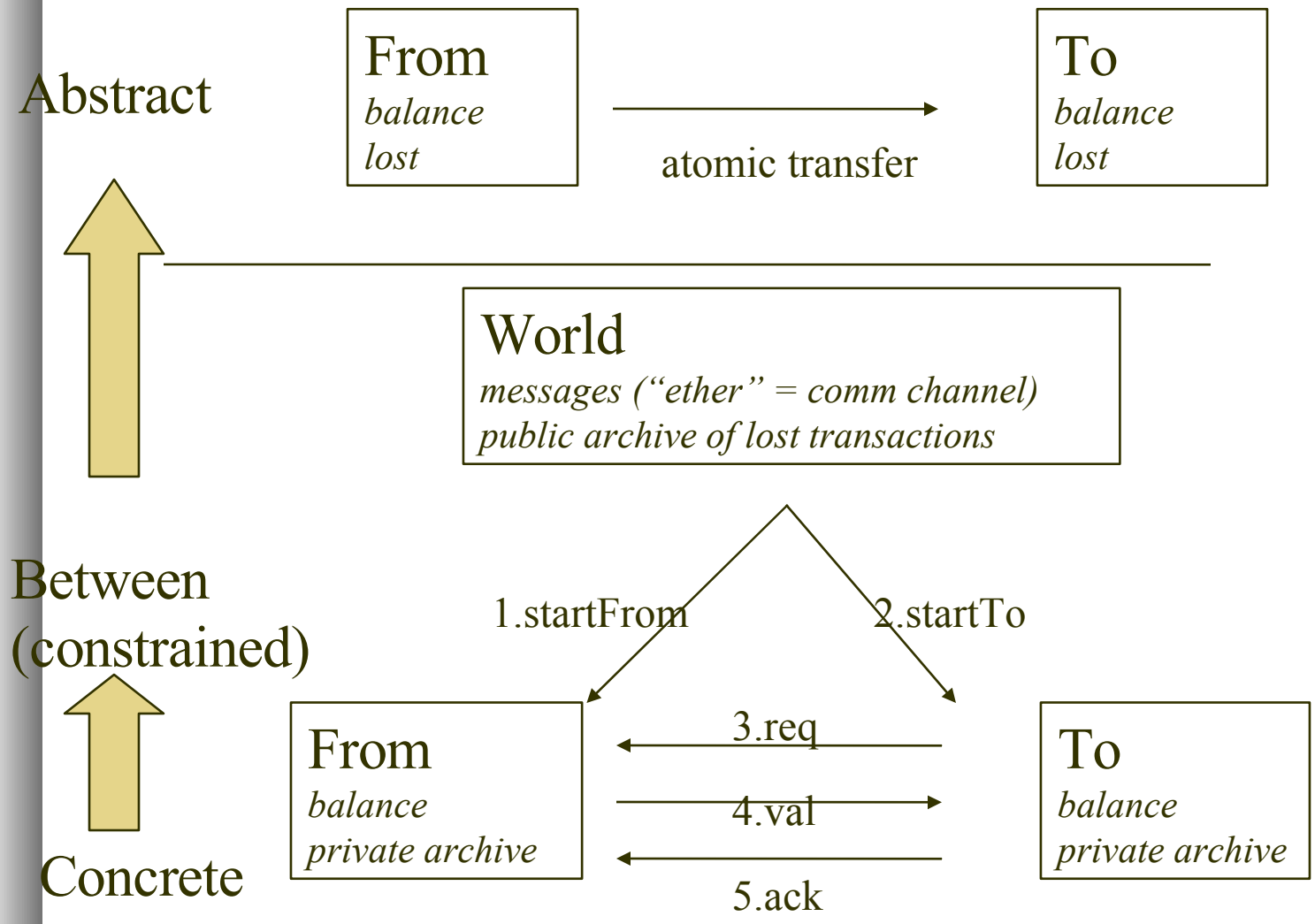
Daniel Jackson
MIT CSAIL Software Design
Cambridge MA, USA

Status Summary

- Progress :
  - Z spec converted into Alloy modules
  - All refinement theorems checked
- Deadlines :
  - End on August 24th
  - presentation at École Normale Supérieure (Paris, France) on September 20th

Principle

Total balances not increasing
Total balances and lost constant

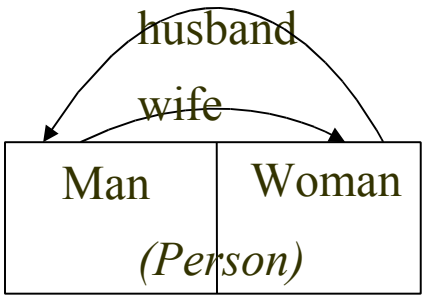


Outline

- Alloy Principles
- Mondex in Alloy : General Method
- Technical issues
- Conclusions

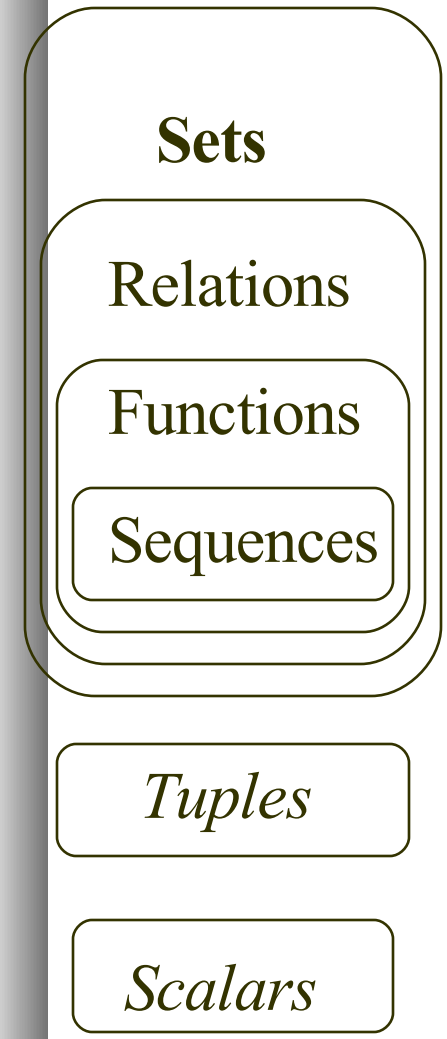
Alloy Spec Language & Logic

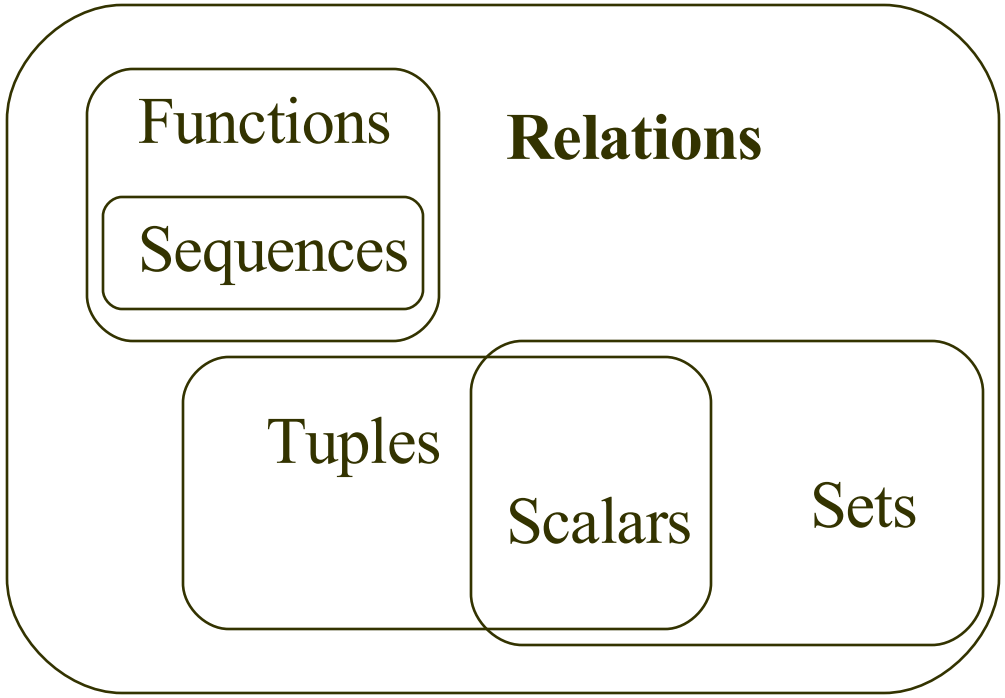
- Typed and modular specification language
- Sets and relations
  - Signatures define particular (“basic”) sets and relations
    - Can be abstract, extended (“inheritance” as in Java)
      - Typing, overloading, modularity
      - quite like Z schema extensions
  - Specification can be constrained
- First order logic + relational calculus
  - Relational operators : union, inter, diff, *join*
- *Everything is finite*




```
abstract sig Person {}
sig Man extends Person {wife:set Woman}
sig Woman extends Person {husband:set Man}

fact Constraint {
   all m:Man |
   some m.wife implies m.wife.husband = m
   all w:Woman |
   some w.husband implies w.husband.wife = w
}
```

Alloy relations vs. Z sets



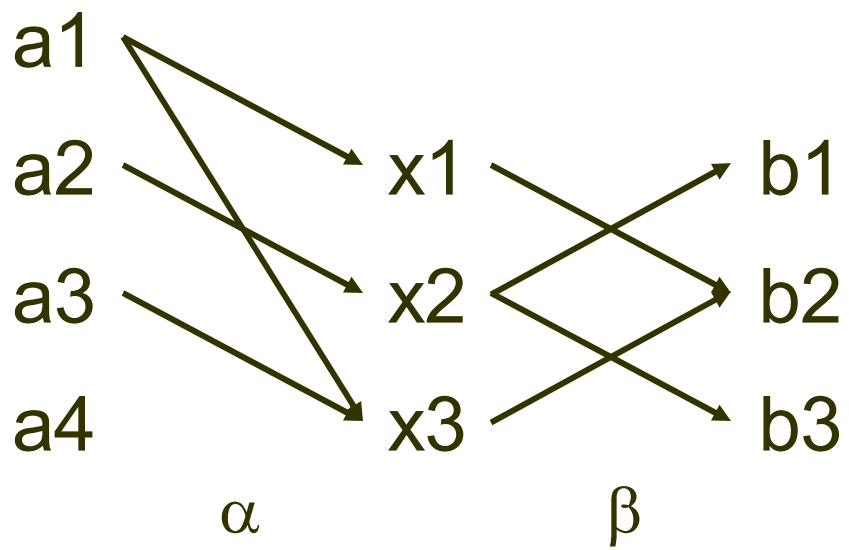


Alloy

- sets are unary relations
- scalars are singletons

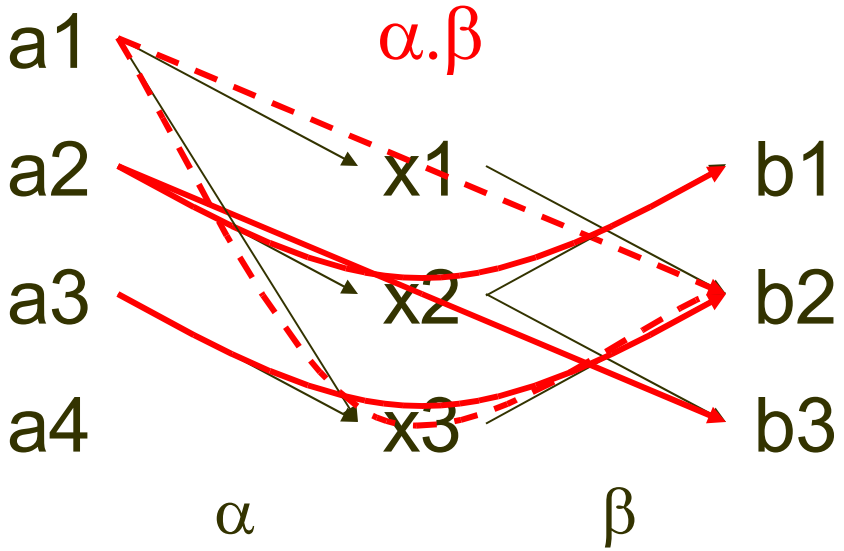
Joining relations (.)

- Let α and β be two relations
  - `sig A {alpha : set X}`
  - `sig X {beta : set B}`
  - `sig B`



Joining relations (.)

- Let α and β be two relations
- so we define $\alpha.\beta$ the *joined relation*
  - *Cf. database* $\triangleright\triangleleft$
- *We may write* `a2.(alpha.beta)=b1+b3` *, it is the same join operator because :*
  - *sets are unary relations*
  - *scalars are singletons*



Alloy Analyzer, a Model Finder

- Specification Analysis by Model Finding
  - "Run" predicate: find example
  - Check assertion: find counterexample
  - Alloy internally converts modules to SAT formula
- "Scope" required : bounded finite models
  - Number of objects for each signature
  - Can show theorems hold in *specified scope*

```
pred Married (p:Person) {some p.(wife+husband)}

pred Example () {some p:Person|Married(p)}
run Example for 18 Man, 1 Woman

assert Theorem {
   all p:Person|lone p.(wife+husband)
   all p,q:Person|p.husband=q iff q.wife=p }
check Theorem for 7
```

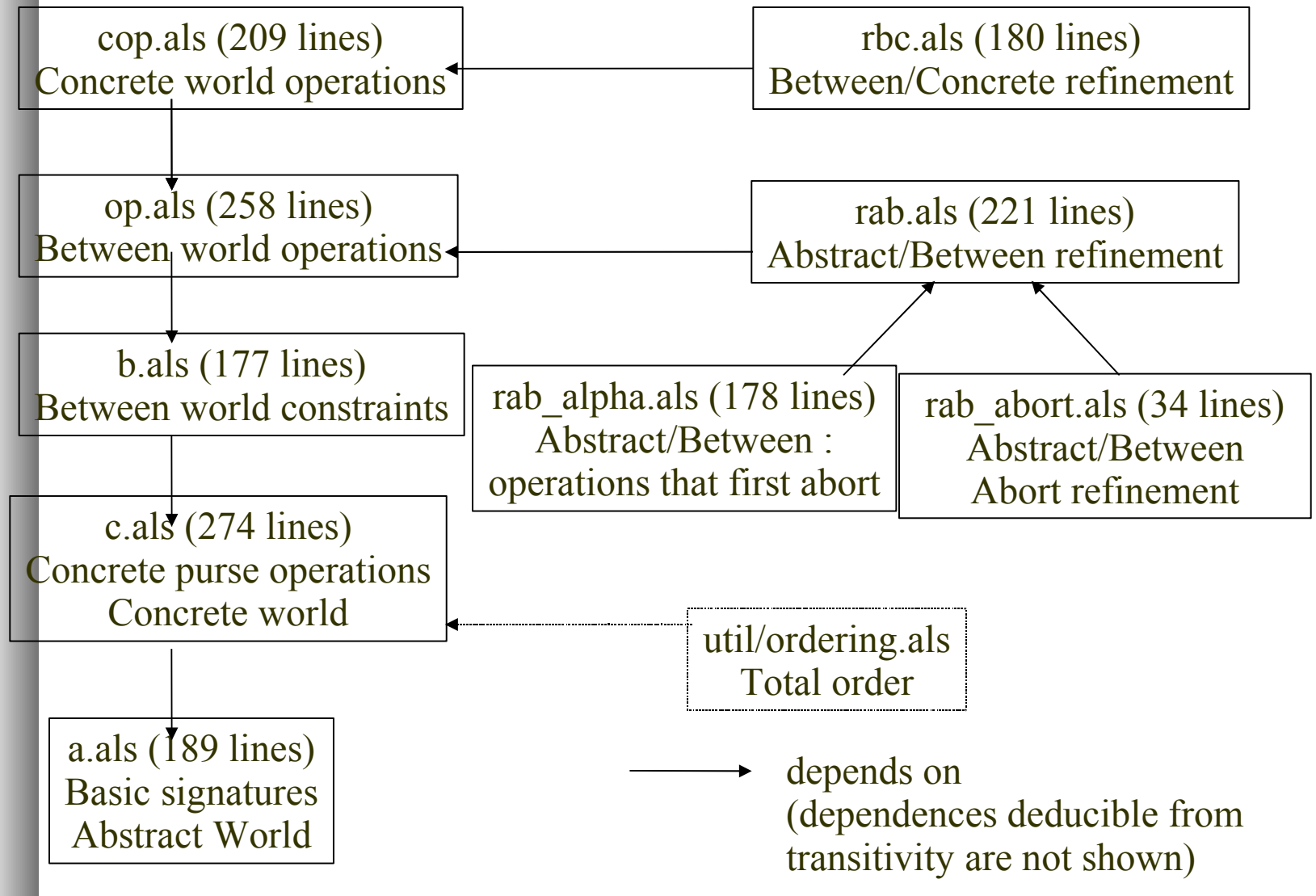
Outline

- Alloy Principles
- Mondex in Alloy : General Method
- Technical Issues
- Conclusions

Bugs found in Z Specification

- Missing constraints
  - 2 ConPurse constraints
  - Avoid ConPurse holding “foreign” pdAuth when in epv/epa
    - Constraint analogous to existing one for epr
- Wrong proof step
  - Proof splitting for A/B Abort
  - Wrong assumption made by informal comment

Spec modules outline



Almost everything represented

- Alloy modules very close to Z specification
  - *Representation* size is comparable
  - Alloy Proof size is negligible
    - Actually no proof details in Alloy modules
  - Quite quick to write (< 1 month)
- Only changes :
  - Integer representation
  - Unable to express infiniteness in Alloy
    - finiteness properties ignored
  - CLEAR code
    - quantify over CLEAR codes instead of their corresponding sets of PayDetails
- Enforces 1st order quantifications

Safety Check : Initial states

- Only case where “run” a predicate
  - ask Alloy to build one model with initial state
  - You may demand further constraints to see what happens (e.g. some purses)
  - No big scope required
    - if example at scope 5, *a fortiori* at bigger scope

```
pred AbInitState (a:AbWorld) { ... }
pred A821 () {some a:AbWorld | AbInitState(a)}

pred A821bis() {
some a:AbWorld {
    AbInitState (a)
    some a.abAuthPurse
}}

run A821 for 5
run A821bis for 5
```

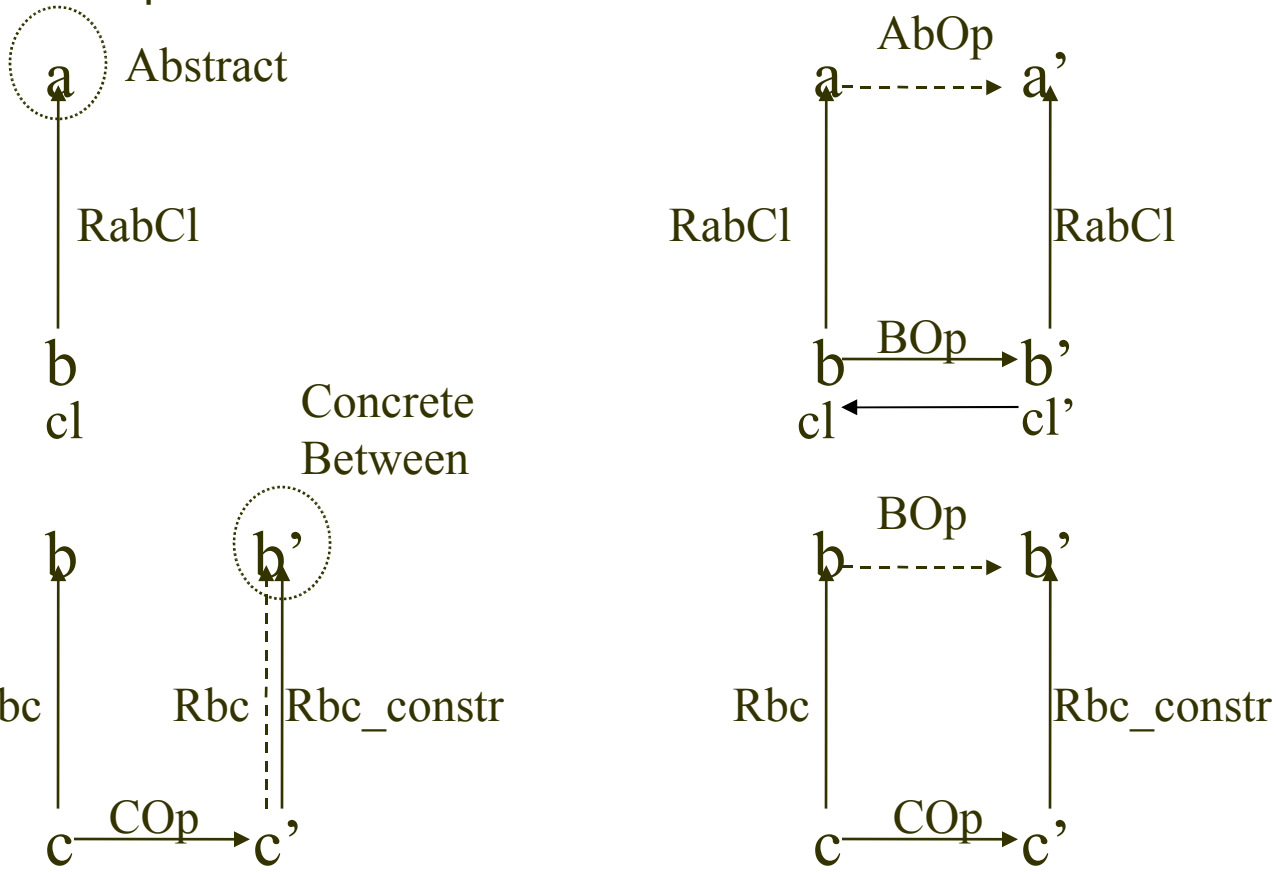
Model consistency : Totality

- Abstract and concrete : check them directly
  - < 1 hour with Berkmin (Abstract) or Mchaff (Concrete)
- Between :
  - Direct checking needs to check the 15 constraints
  - But any operation may do nothing
  - So, check that Op(x, x) holds
    - Explicitly provide witness for ∃x', Op(x, x')
    - Checks faster : <1 hour each with Berkmin

```
assert C832_val
{all c:ConWorld, m_in:MESSAGE, name_in:NAME |
some c':ConWorld, m_out:MESSAGE| Cval(c,c',name_in,m_in,m_out)}
check C832_val for 10

assert B832_val
{all b,m_in,name_in:NAME|
some m_out:MESSAGE| Val(b,b,name_in,m_in,m_out)}
check B832_val for 10
```

Refinements : checking method

- Follow Z spec strategy (A/B backwards, B/C forwards)
  - But separate existence and refinement



- Rbc_constr : equality predicates (explicit “construction”)
  - Not necessary for RabCl (already in this form)

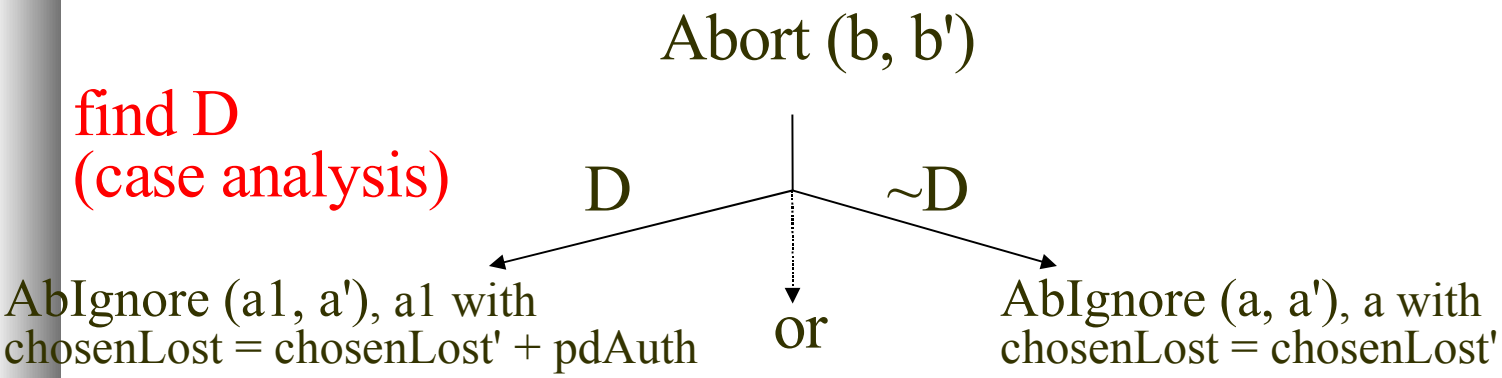
Abstract/Between : RabCl

- Abstraction relation RabCl already gives a construction (written as equalities) depending on ChosenLost (prophecy variable)
- Quite long to check (scope of 8 takes >26000s with Berkmin)

```
sig ChosenLost {pd: PayDetails}
fun RabBalance (b:ConWorld, cl:set PayDetails, n:NAME) : set Coin {…}
fun RabLost (b:ConWorld, cl:set PayDetails, n:NAME) : set Coin {…}

pred Rab (a:AbWorld0, b:ConWorld, cl:set PayDetails){
all n:NAME {
     lone n.(a.abAuthPurse)
     n in NAME.(b.conAuthPurse) implies {
          some n.(a.abAuthPurse)
          n.(a.abAuthPurse).balance = RabBalance(b,cl,n)
          n.(a.abAuthPurse).lost = RabLost(b,cl,n)
}}}

assert rab_ex {
     all b:ConWorld, cl:ChosenLost, a:AbWorld0 |
     RabCl (a, b, cl.pd)
     implies Abstract (a.abAuthPurse)
}

check rab_ex for 8
```

Abort

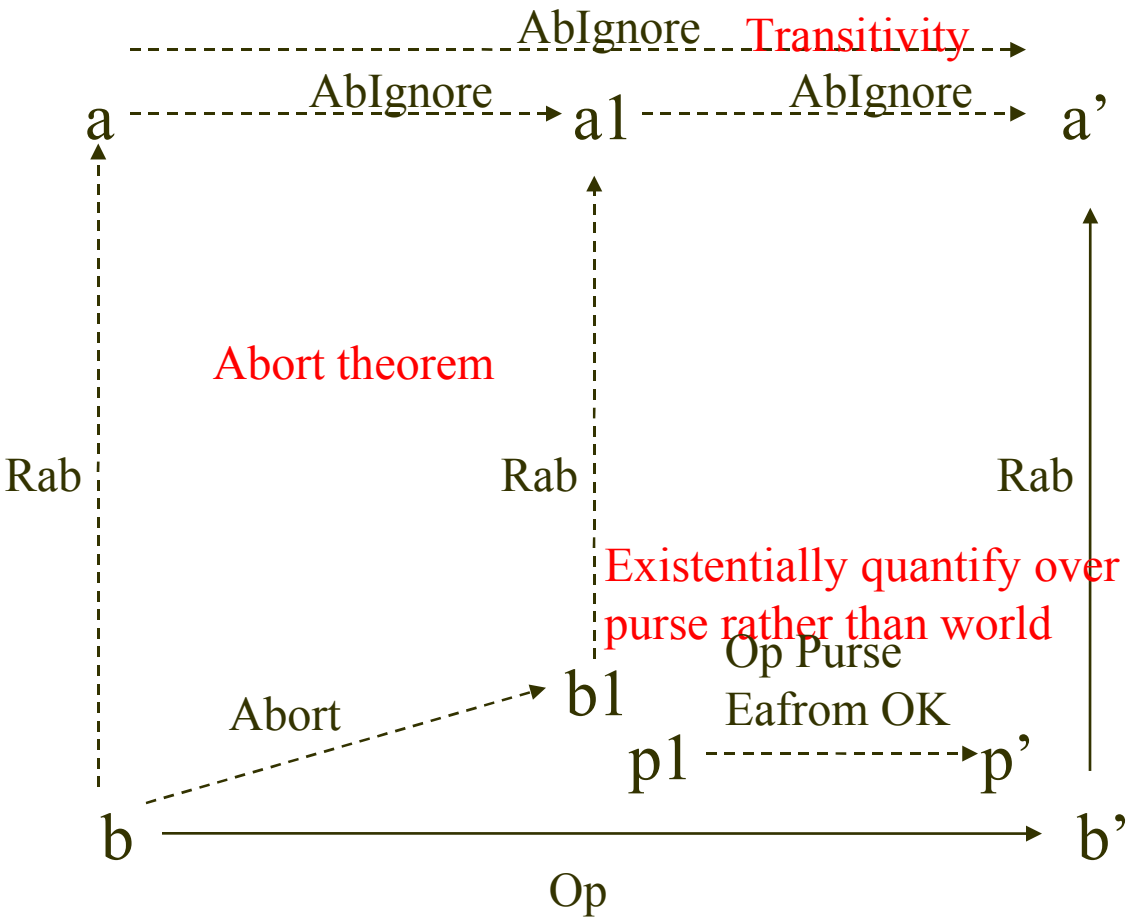
- Most difficult theorem
  - Direct attempt does not terminate after 4 days with Siege_v4
  - So, requires one step towards proof details



  - Z spec suggests D : splitting proof whether pdAuth in maybeLost
  - This splitting is wrong !
    - found counterexample where aborting purse is not the to purse expecting val (was actually the from purse)
  - Right splitting condition is D : aborting purse in epv
    - Works well and terminates in <30000s

Operations that first abort

- StartFrom, StartTo and exception logging
  - conjunct with ~Abort
  - scope of 8 takes <24000s with Siege_v4

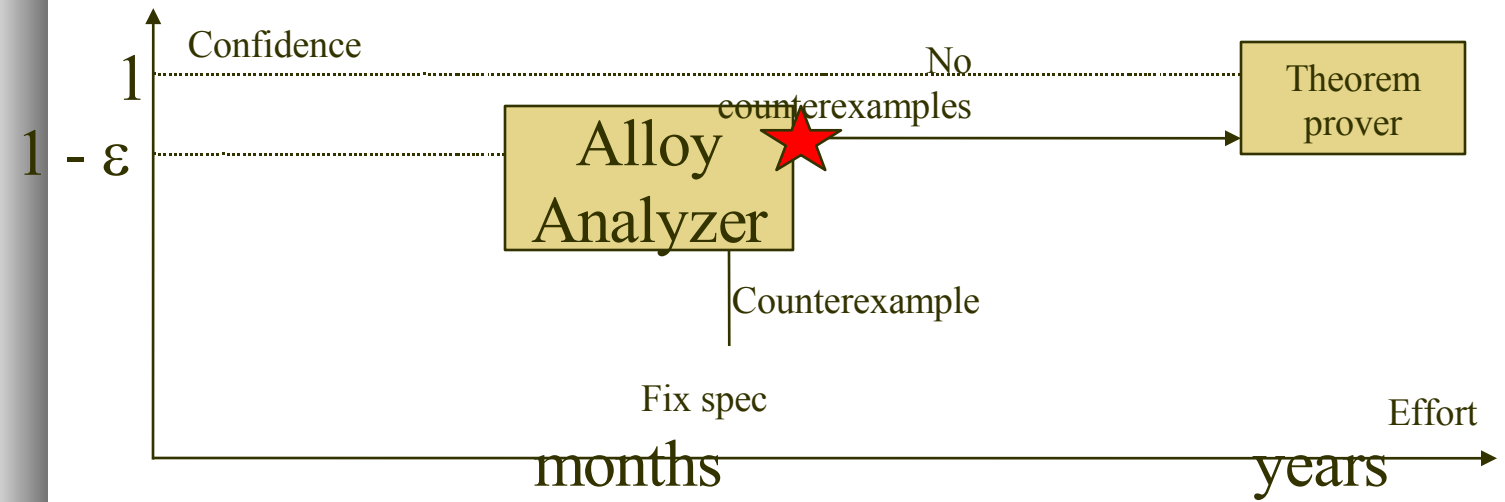


ConPurse missing constraints

- 2 constraints missing in Z spec
  - found while checking Between/Concrete existence
    - $status = epv \Rightarrow pdAuth.to = name$
      $status = epa \Rightarrow pdAuth.from = name$
  - Found counterexample for which purse holds “foreign” `pdAuth`
  - Even though never happens in *full* sequence

Alloy's Approach Summary

- Refinement checks with model finding
  - Try to find c, c', a, a' such that Rac(a, c) & Rac(a', c') & COp(c, c') hold but not AOp(a, a')
- Original approach
  - Quite high confidence level
  - Not as high as theorem proving
  - but much cheaper !



Outline

- Alloy Principles
- Mondex in Alloy : General Method
- Technical Issues
- Conclusions

Integers in Alloy

- Integers in Alloy are heavy
  - Builds boolean circuits for +, <
  - Expensive operations
- So, avoid them
  - Not all properties of N used
  - Determine which
  - Pick most lightweight repr that works

Representing SEQNO

- Avoid integers in Alloy
- SEQNO just requires total order
  - No operations
  - Even no successor
- Simply use Alloy's ordering module
  - Exploit built-in symmetry breaking too

Representing amounts

- Avoid integers in Alloy
  - Distributed sum available, but too expensive
- Solution : sets of coins
  - Due to Emina Torlak & Derek Rayside

Z	Alloy
Integers	Sets of coins
Equality	Set equality
Ordering	Set inclusion
Sum	Set union
Difference	Set difference

- OK, because no comparison between purses
  - Globally : coins between whole worlds
  - Locally : between a purse balance & a payment
- Add constraints to avoid coin sharing

Concrete purse : Z and Alloy

[NAME]

ConPurse

balance : **N**
exLog : **P** PayDetails
name : NAME
nextSeqNo : **N**
pdAuth : PayDetails
status : STATUS

$\forall$ pd : exLog • name $\in$ {pd.from, pd.to}

status = epr $\Rightarrow$
name = pdAuth.from
$\wedge$ pdAuth.value $\leq$ balance
$\wedge$ pdAuth.fromSeqNo < nextSeqNo

status = epv $\Rightarrow$
pdAuth.toSeqNo < nextSeqNo

status = epa $\Rightarrow$
pdAuth.fromSeqNo < nextSeqNo

```
sig NAME {}
sig Coin, SEQNO {}
open util/ordering[SEQNO] as seqord

sig ConPurse {
      balance : set Coin, exLog : set PayDetails,
      name : NAME, nextSeqNo : SEQNO,
      pdAuth : set PayDetails, status : STATUS
}

fact {all c:ConPurse {

all p:PayDetails|p in c.exLog implies name in p.from+p.to

c.status = epr implies {
      name=c.pdAuth.from
      c.pdAuth.value in c.balance
      seqord/lt (c.pdAuth.fromSeqNo, c.nextSeqNo)
}

c.status = epv implies {
      name=c.pdAuth.to
      seqord/lt (c.pdAuth.toSeqNo, c.nextSeqNo)
}

c.status = epa implies {
      name=c.pdAuth.from
      seqord/lt (c.pdAuth.fromSeqNo, c.nextSeqNo)
}

no c.balance & c.exLog.value

}}
```

Signatures are not records

- Z : schemas are records
- Alloy : signatures define atomic objects
  - Objects have an *identity*
    - Notion does not exist in Z
  - Suitable for names, coins
- Two objects with same field values may be distinct
  - Solution : impose equality constraint

```
fact {
    no disj c1,c2:ConPurse {
        c1.balance=c2.balance and c1.exLog=c2.exLog
        c1.name=c2.name and c1.nextSeqNo=c2.nextSeqNo
        c1.pdAuth=c2.pdAuth and c1.status=c2.status
    }
}
```

Existential issue

- Can't guarantee object exists for every combination of field values
  - Could axiomatize with constraints
  - But would dramatically increase scope
- Solution : (cf. RabCl)
  - Instead of E, construct explicit witness
  - all c, c', a | some a' | P (c, c', a, a')
  - all c, c', a | let a' = F(c, c', a) | P(c, c', a, a')

Choosing scopes

- Must be enough for quantifications
- Started with 10
  - worked fine with Abstract theorems
  - too long for more complex theorems
    - SAT solver crashed for refinement checks
  - so grow scope incrementally
- Achieved scope of 8 for most theorems eventually
  - but smaller scope is complete for Worlds

Scope	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Between Ignore sanity check	135	715	2714	6286	**15383**		
explicit post-state					*31*		***54***
Abstract/Between existence	52	458	2606	11498	**26690**		
Between/Concrete existence	3537	**22059**					
Siege_v4, restricted World scopes					***55042***		
Between/Concrete StartFrom	18	105	308	848	2526	6309	**13951**

First attempt to check theorems. At that stage they had been checked with Berkmin and without any optimization. *Italics* indicate timing after optimizations.
Time in seconds in function of scope.

Outline

- Alloy Principles
- Mondex in Alloy : General Method
- Technical issues
- Conclusions

General observations

- Modeling
  - Transcribed Z to Alloy very directly
  - May be better to try Alloy idiom?
- High level checking
  - Proof structure not needed: automated
  - Exception: abort splitting
  - Need to provide explicit witness for ∃
- SAT-Solving duration varies
  - From seconds to hours (even days!)
  - Time correlated with theorem importance?

Alloy Limitations

- Alloy is finite
  - Can express unbounded but not infinite models
  - But in practice, world of purses finite
- Alloy Analyzer's analysis is bounded
  - Results valid only on given scope
  - Is scope of 8 enough?
- Reasonable tradeoff for industry?
  - Much less effort than theorem proving

Personal Experience

- Learn Z and Alloy *from scratch*
- Nice :
  - Language easy to understand
    - no Δ/Ξ/graphical issues
  - Though quite close to Z
  - Expressive & smooth relation logic
- Nasty :
  - Signatures are not records
    - Equality & Existential theorems
  - Resource- and time-consuming SAT-Solving
    - Very long time for obvious-looking theorems (easily provable by hand, e.g. Ignore refinements)
    - Perhaps syntactic pre-analysis would help?

Lessons and future work

- Lessons
  - Learn another verification approach
    - Automation does not exclude proof formalism
  - Even though not theorem proving
    - But allows also checking informal comments
  - Discover problems more quickly
- Future work
  - Improve formal model
    - More uniform treatment of existential theorems
    - Experiment with more Alloy-like idiom (eg, objects)
  - Prove or argue small model theorem?
  - Interface Alloy method with others
    - Depends on workshop outcome

Any questions ?

- E-mail addresses
  - ramanana@mit.edu Tahina Ramananandro
  - dnj@mit.edu Daniel Jackson
- Alloy modules available at :
  - http://www.eleves.ens.fr/~ramanana/work/mondex
- Alloy Website :
  - http://alloy.mit.edu