Coverage of OCL Operation Specifications and Invariants

Mathias Soeken, **Julia Seiter**, Rolf Drechsler

Institute of Computer Science
University of Bremen
www.informatik.uni-bremen.de/agra

July 24th 2015

@agra_uni_bremen

Outline

1. Motivation and example

Outline

1. Motivation and example
2. Coverage in the design flow

Outline

1. Motivation and example
2. Coverage in the design flow
3. Coverage at the Formal Specification Level

Outline

1. Motivation and example
2. Coverage in the design flow
3. Coverage at the Formal Specification Level
4. Implementation: USE plugin

Outline

1. Motivation and example
2. Coverage in the design flow
3. Coverage at the Formal Specification Level
4. Implementation: USE plugin
5. Experimental evaluation

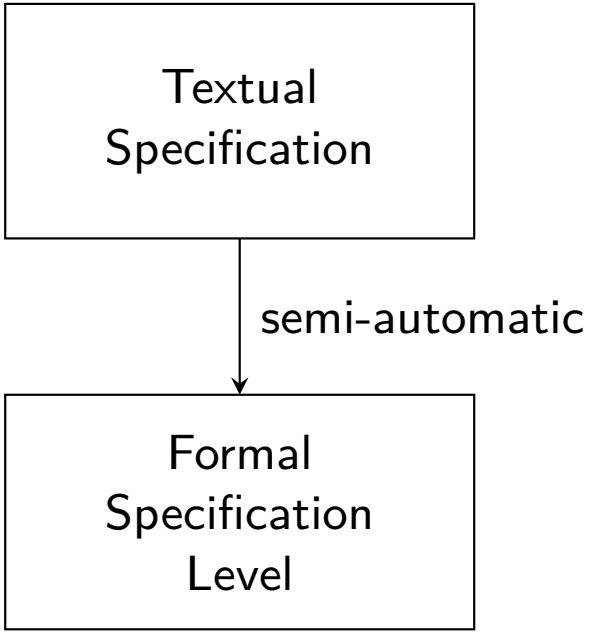
Outline

1. Motivation and example
2. Coverage in the design flow
3. Coverage at the Formal Specification Level
4. Implementation: USE plugin
5. Experimental evaluation
6. Conclusion

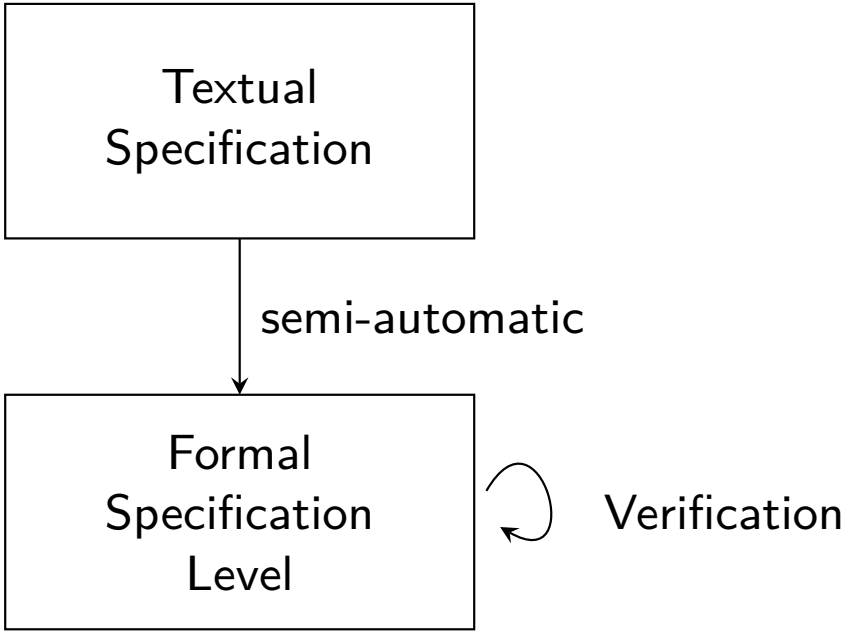
Today's Design Flow

Textual
Specification

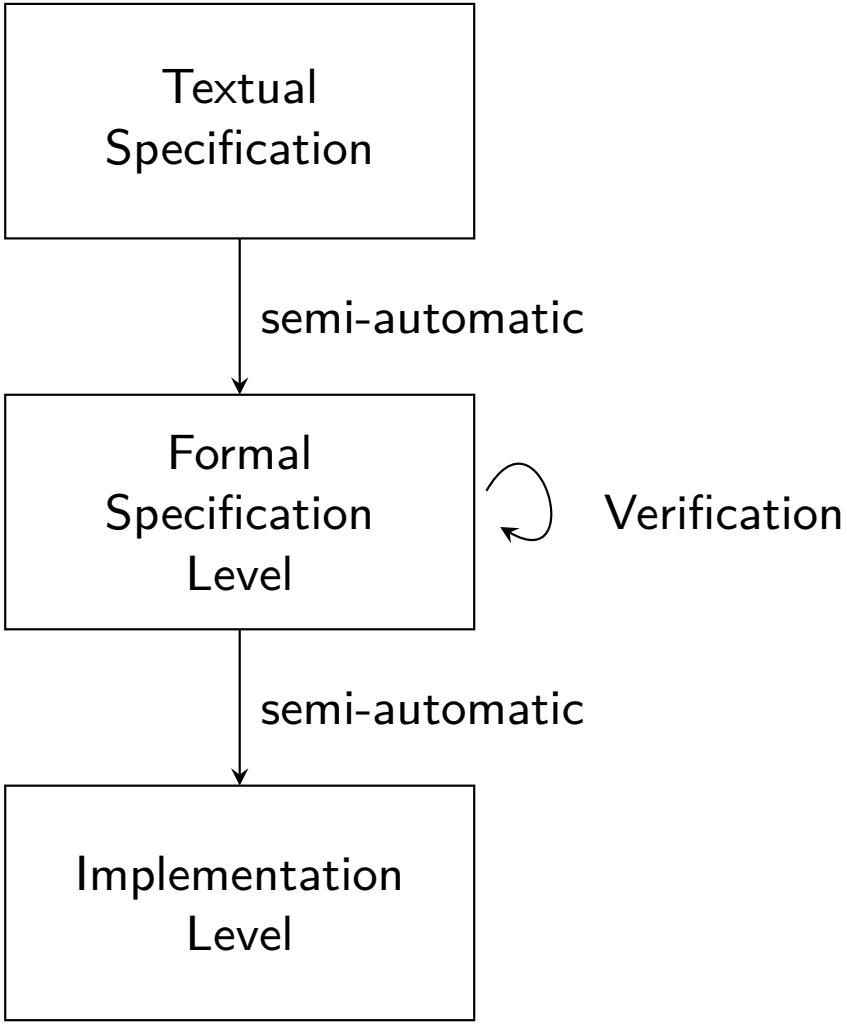
Today's Design Flow



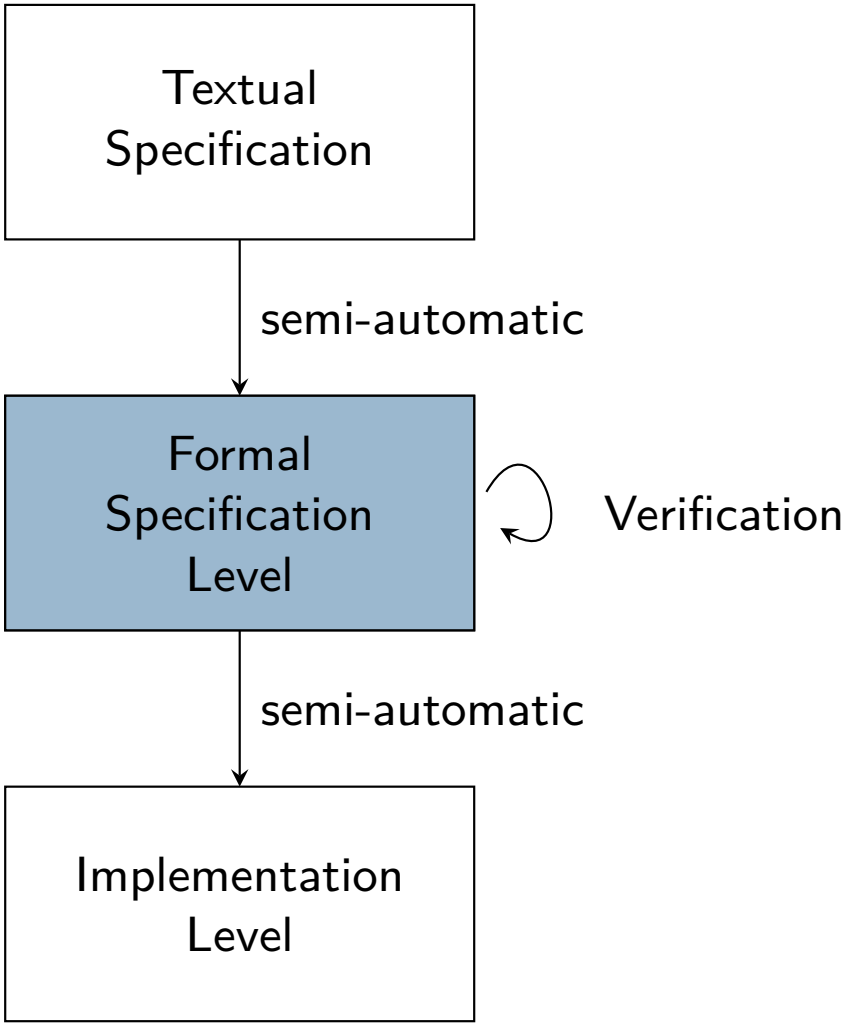
Today's Design Flow



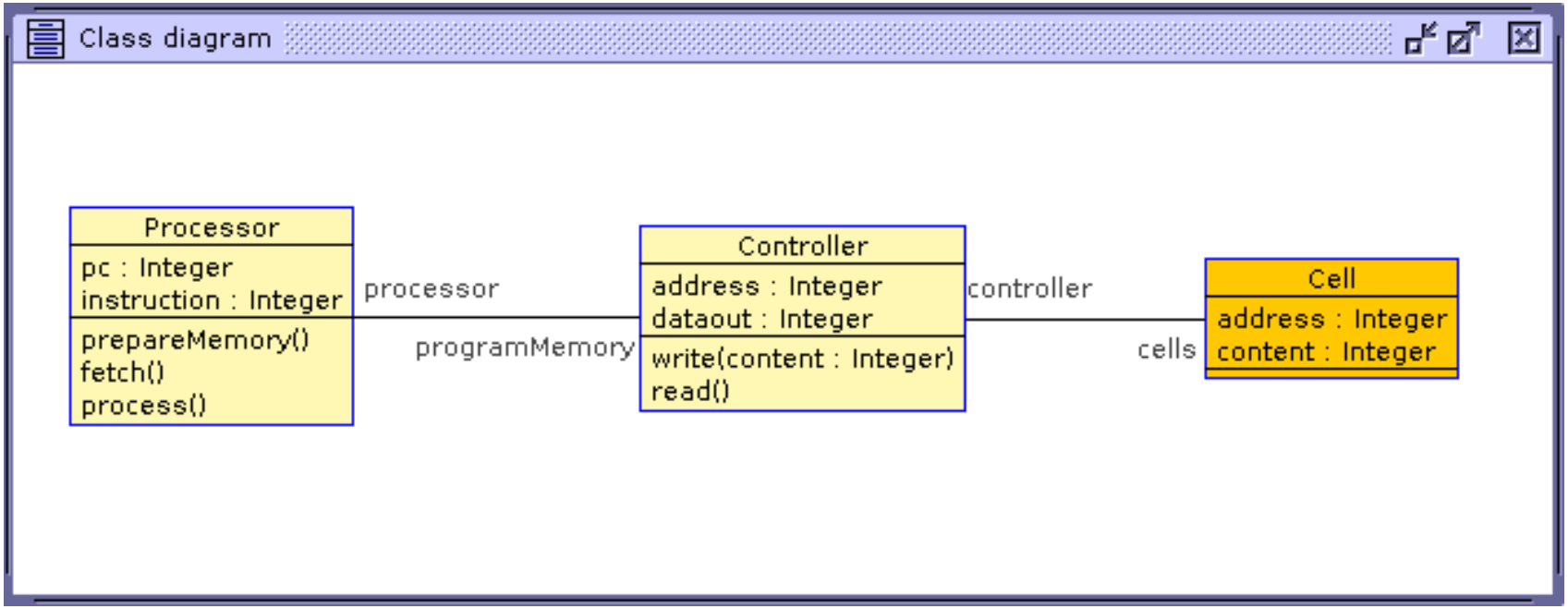
Today's Design Flow



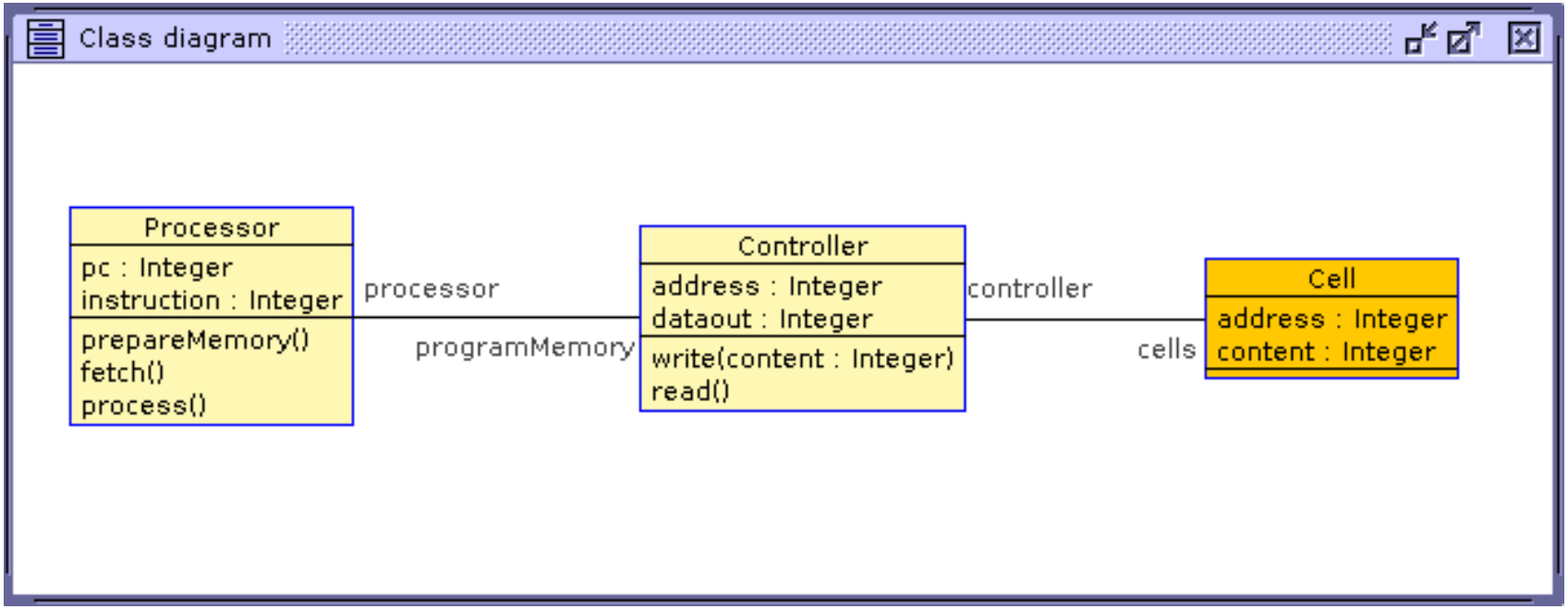
Today's Design Flow



Example: A Memory Controller

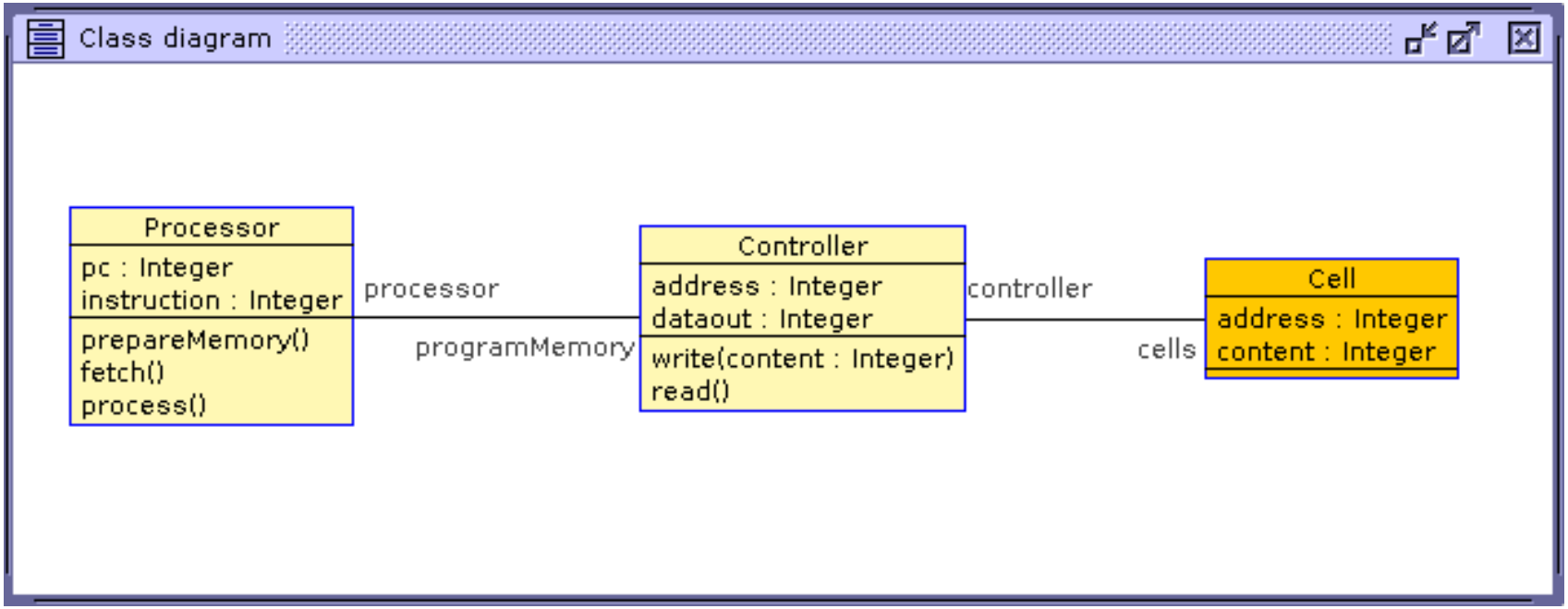


Example: A Memory Controller



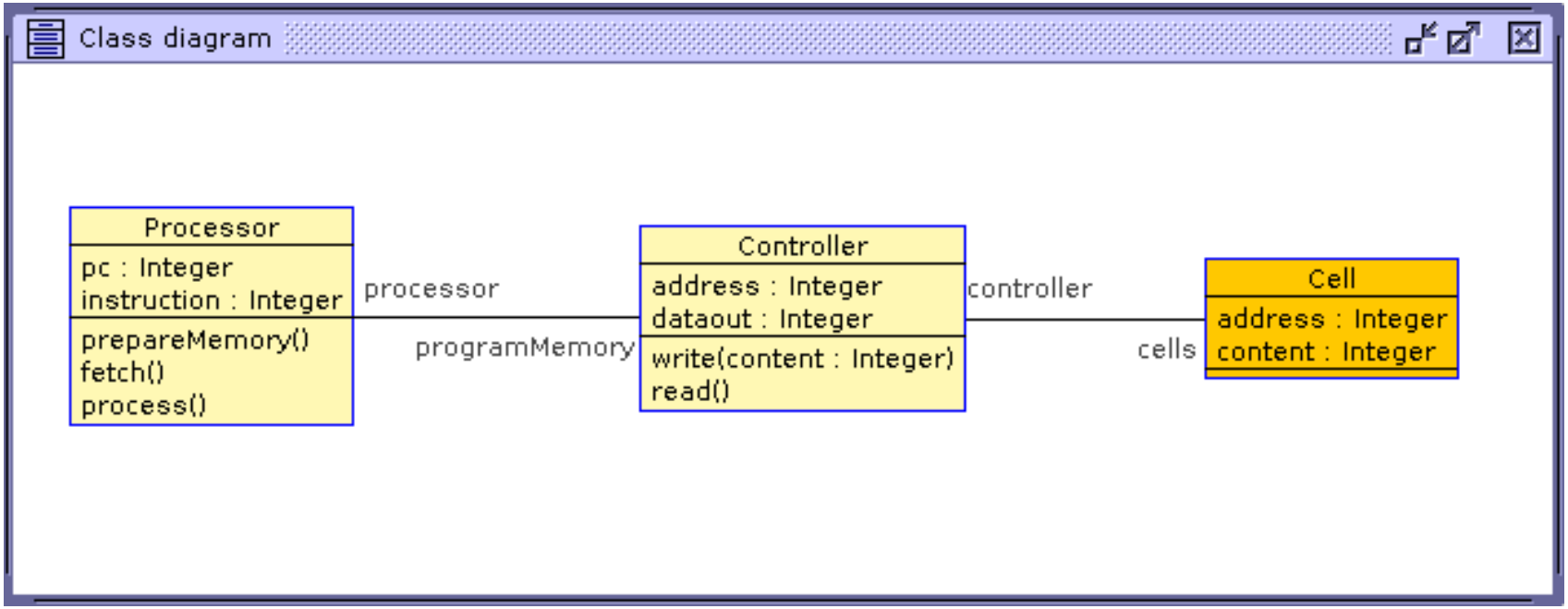
- Consistency

Example: A Memory Controller



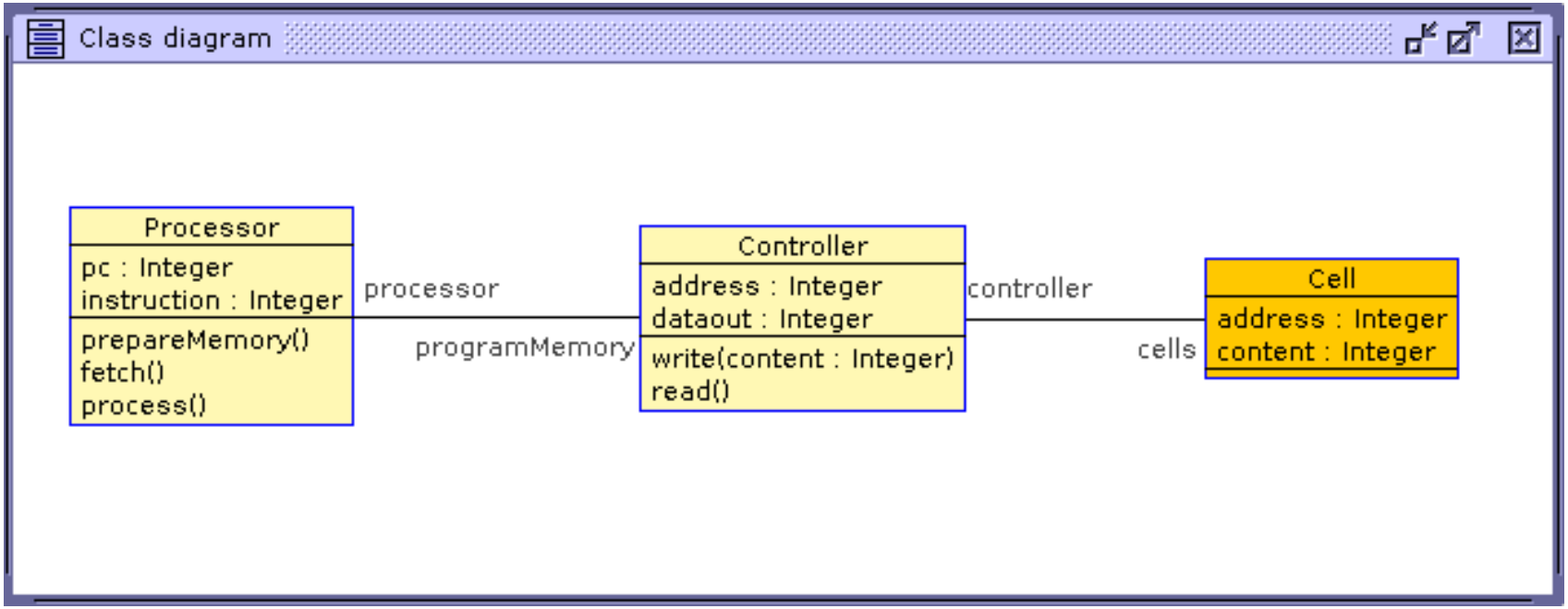
- Consistency
- Executability of operations

Example: A Memory Controller



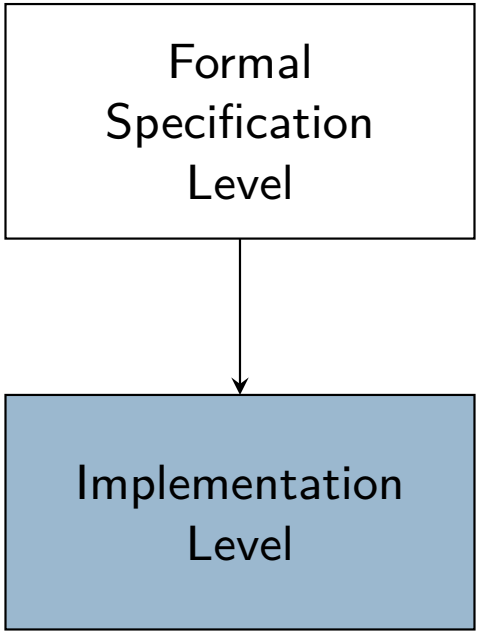
- Consistency
- Executability of operations
- Reachability of a deadlock state

Example: A Memory Controller



- Consistency
- Executability of operations
- Reachability of a deadlock state
- ...

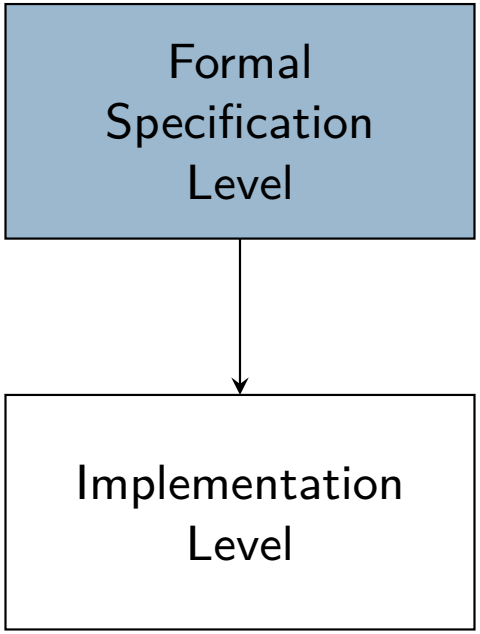
Coverage in the Design Flow



Implementation level:

- Line coverage
- Statement coverage
- Branch/decision coverage
- Path coverage
- Loop coverage

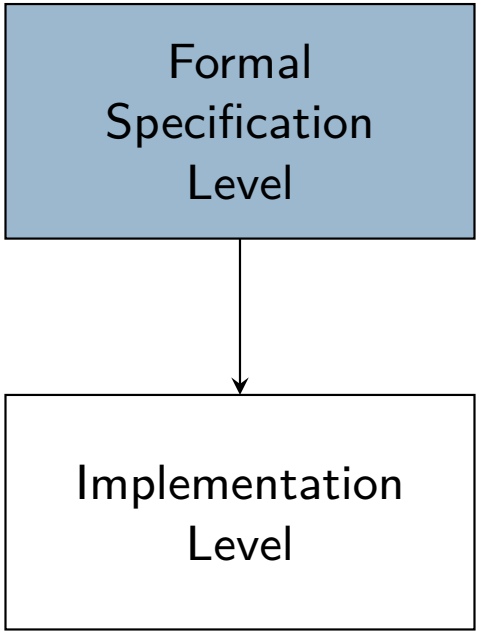
Coverage in the Design Flow



Formal Specification Level:

- Different metrics for different diagrams/model types

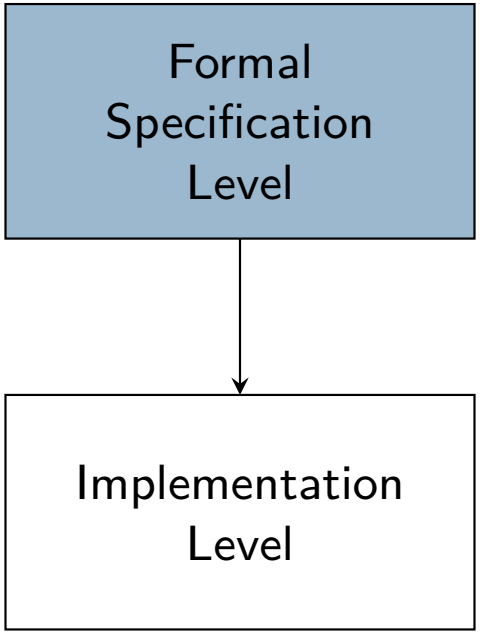
Coverage in the Design Flow



Formal Specification Level:

- Different metrics for different diagrams/model types
- Fewer metrics

Coverage in the Design Flow



Formal Specification Level:

- Different metrics for different diagrams/model types
- Fewer metrics
- Code coverage at the FSL?

Coverage at the FSL

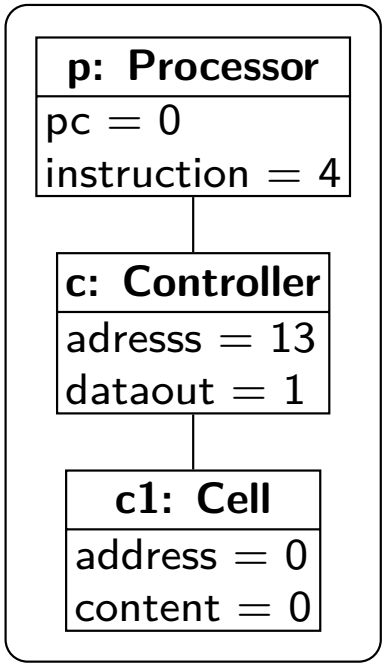
Input: UML class diagram m with OCL constraints,
operation call sequences S

Coverage at the FSL

Input: UML class diagram m with OCL constraints,
operation call sequences S

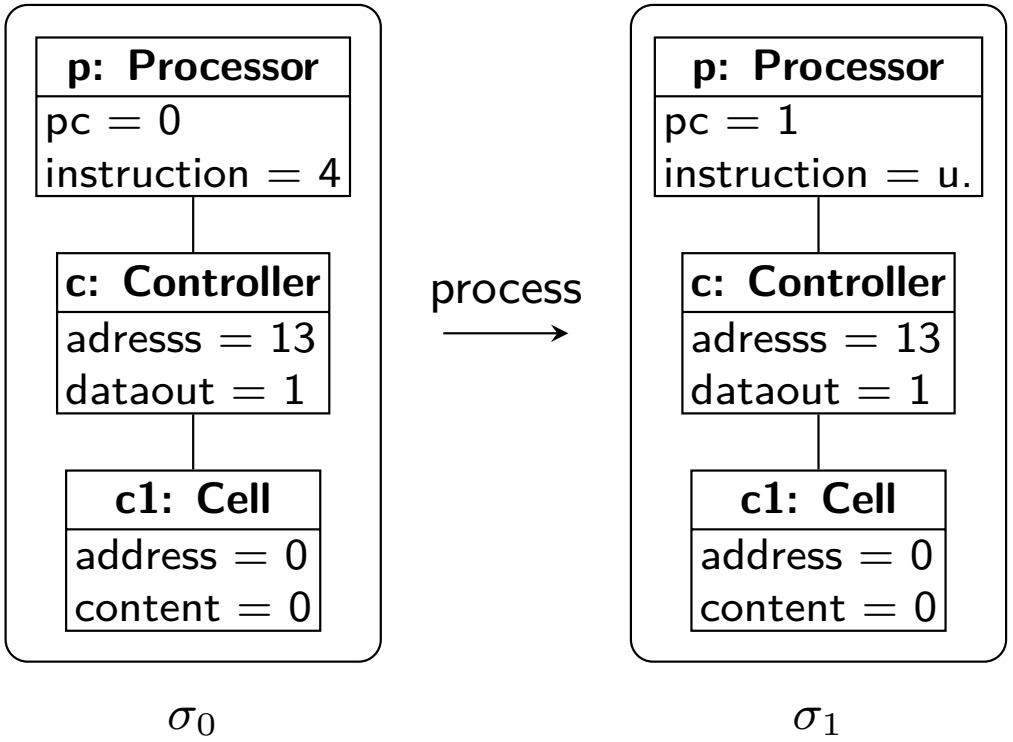
1. **Operation call coverage:** How many operations from m have been called in S?

Operation Call Coverage

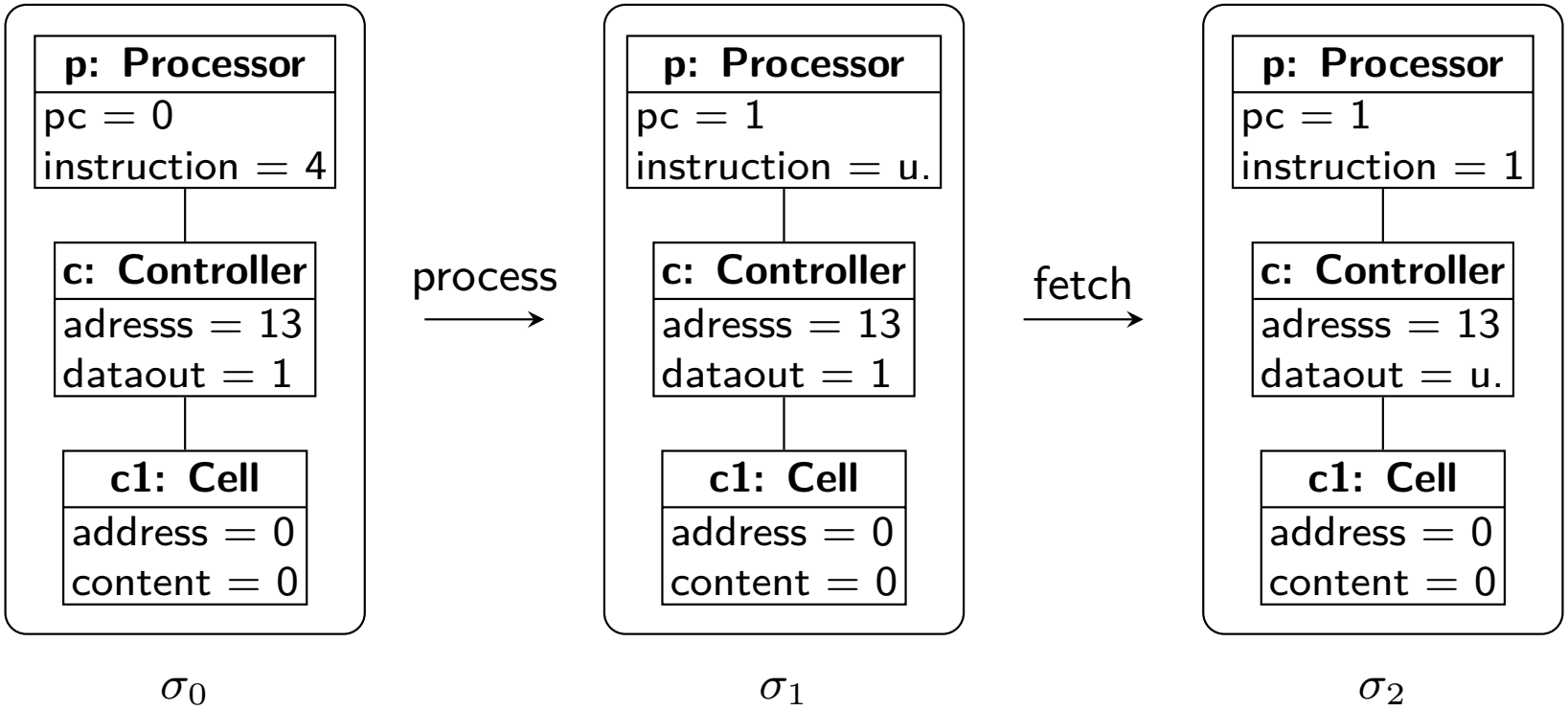


σ_0

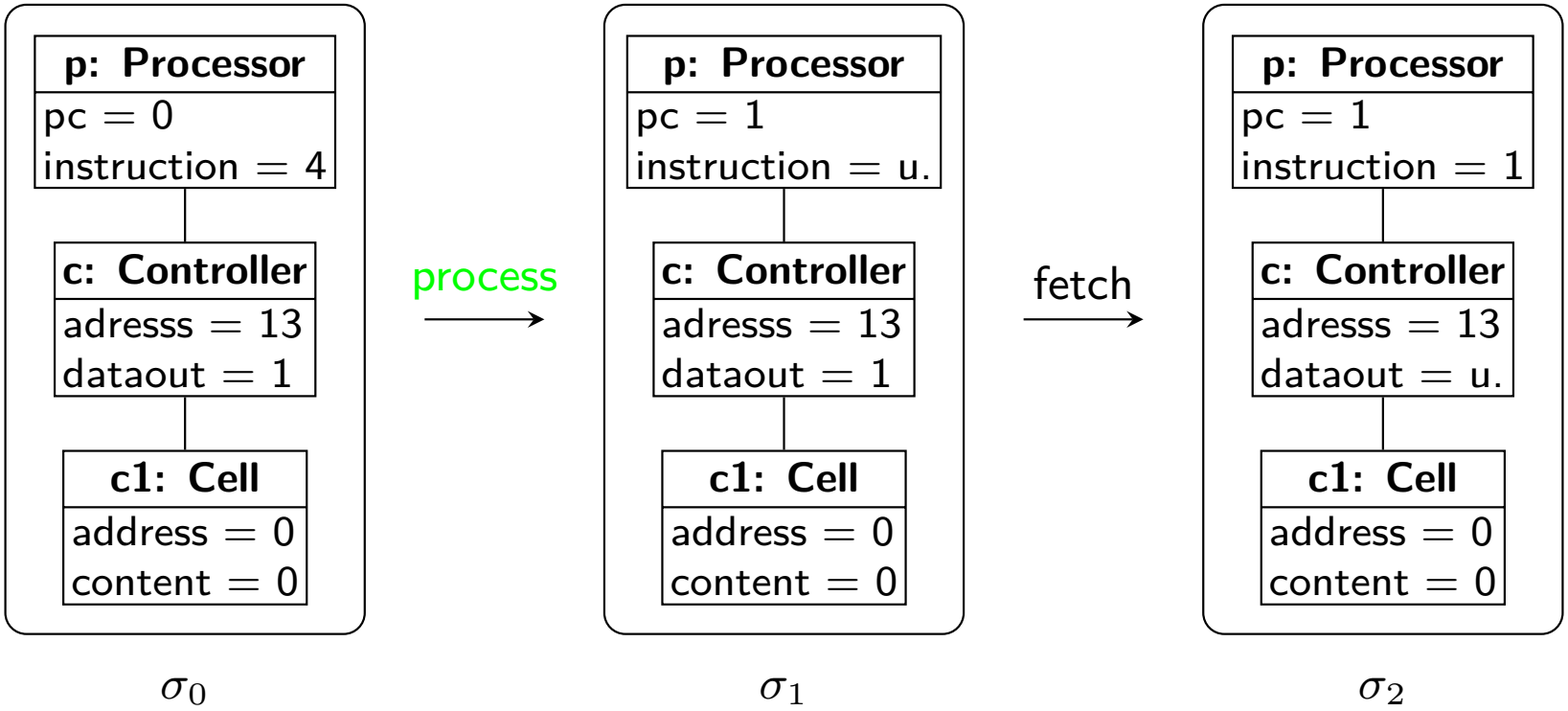
Operation Call Coverage



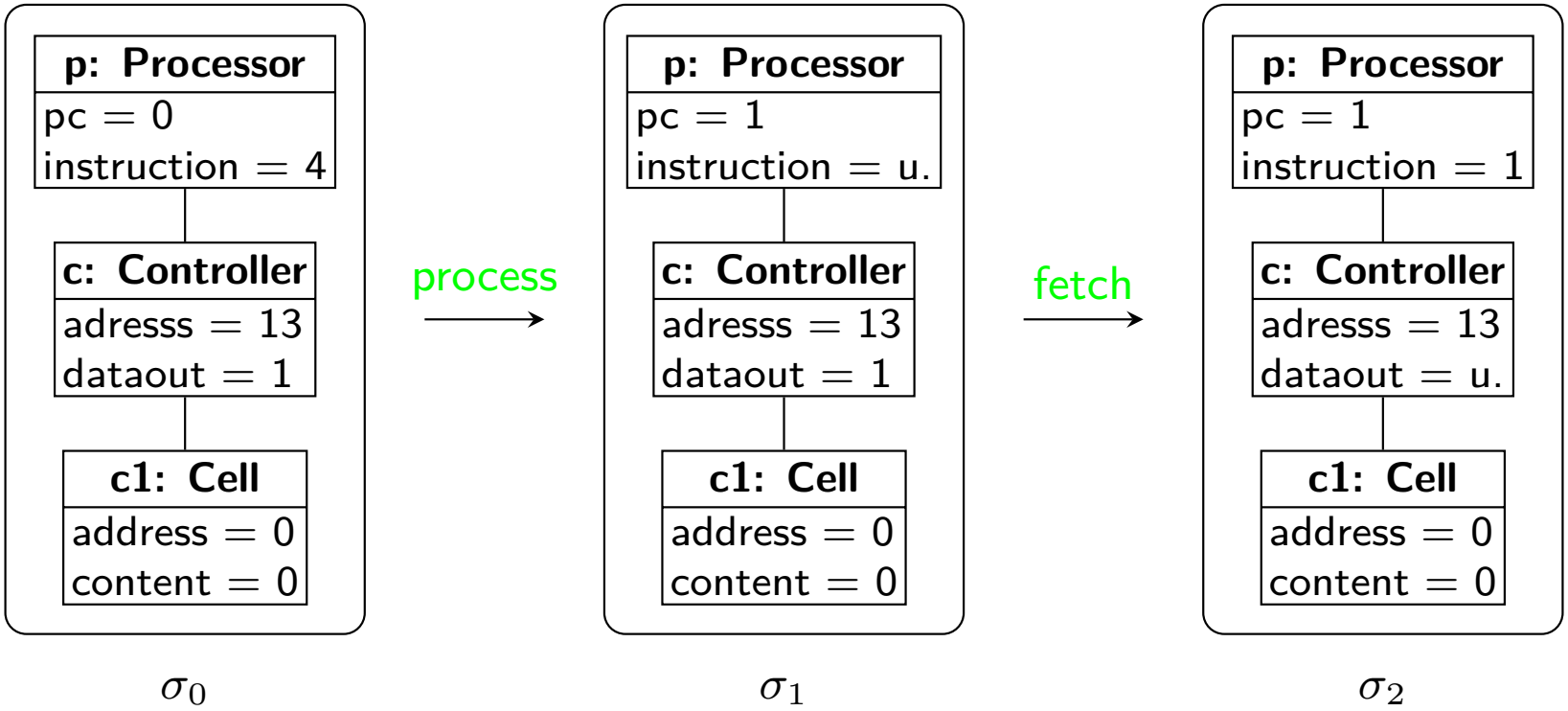
Operation Call Coverage



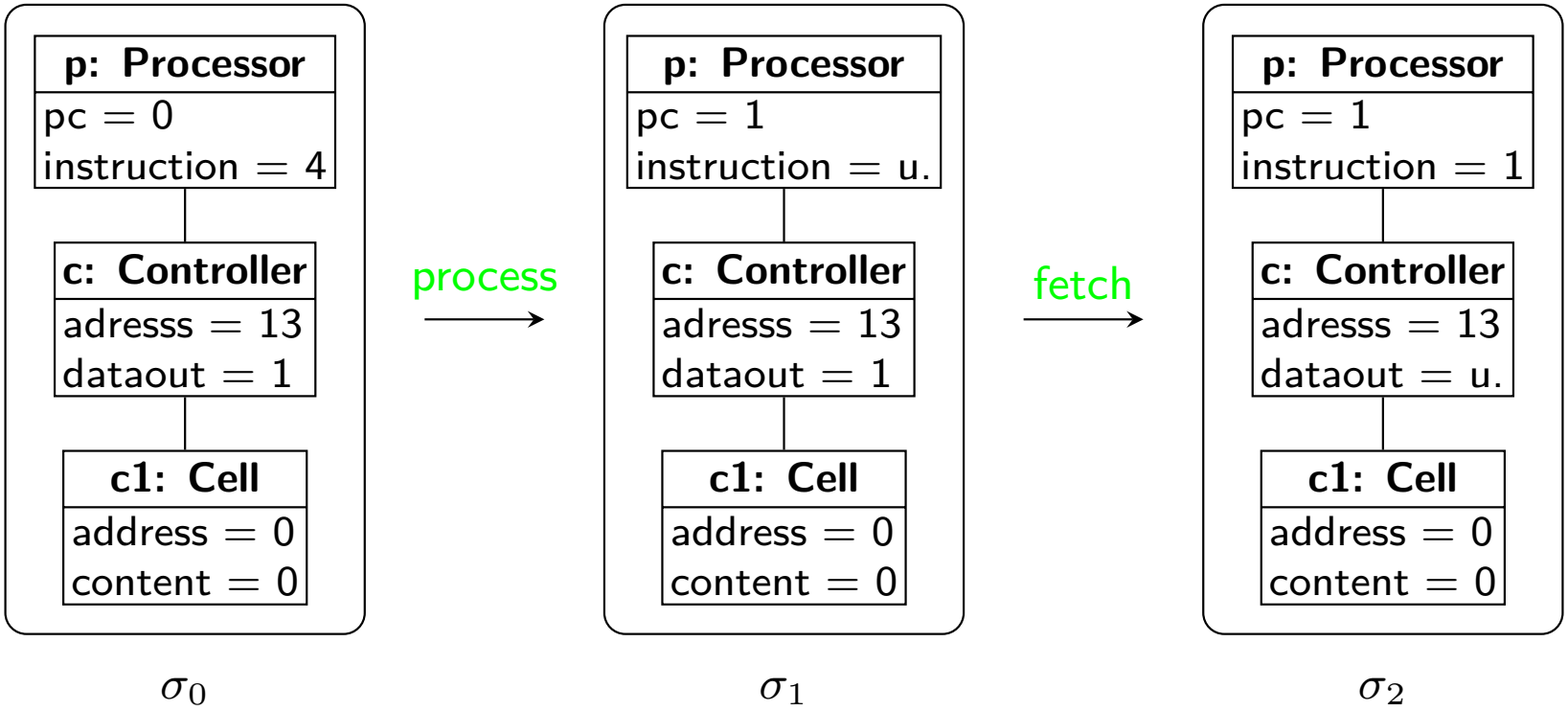
Operation Call Coverage



Operation Call Coverage



Operation Call Coverage



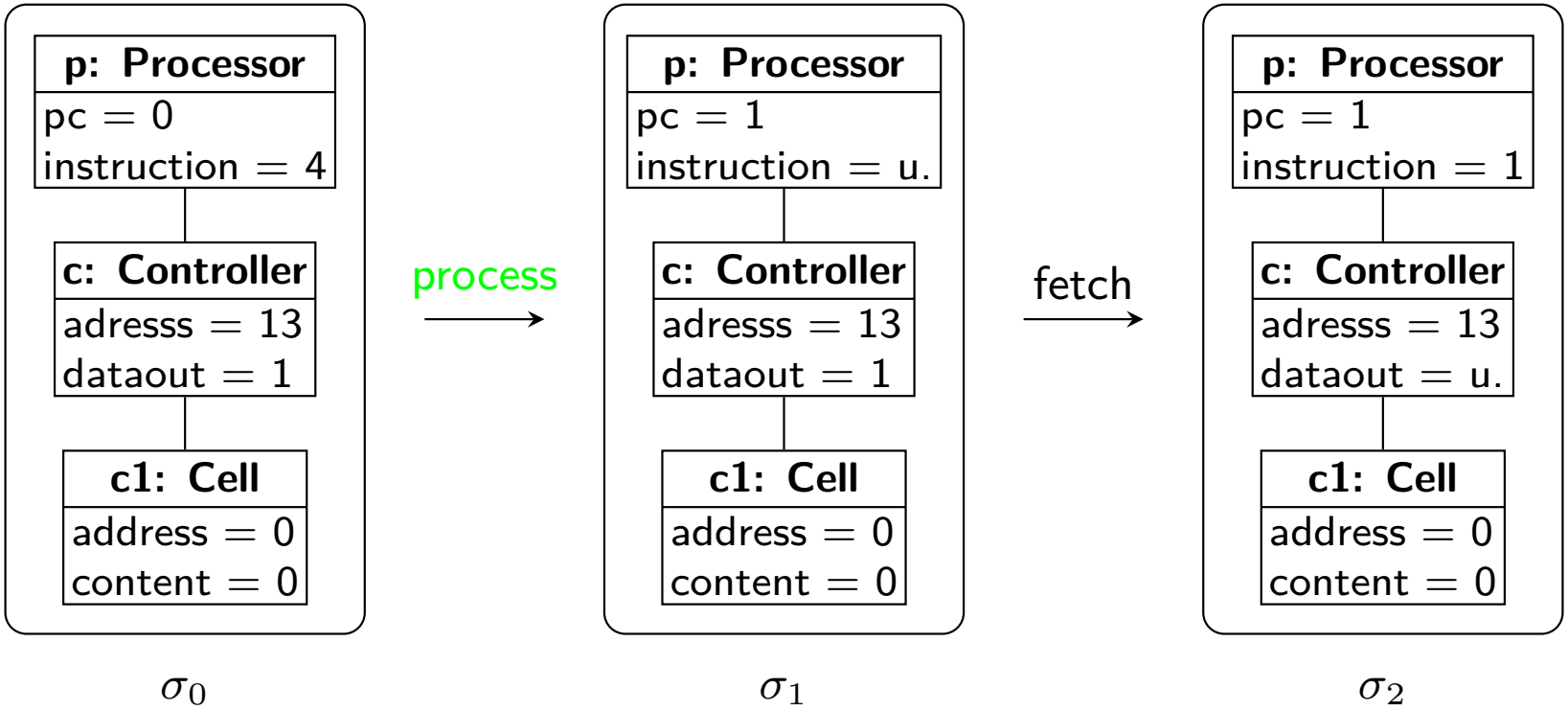
Operation call coverage: 40%

Coverage at the FSL

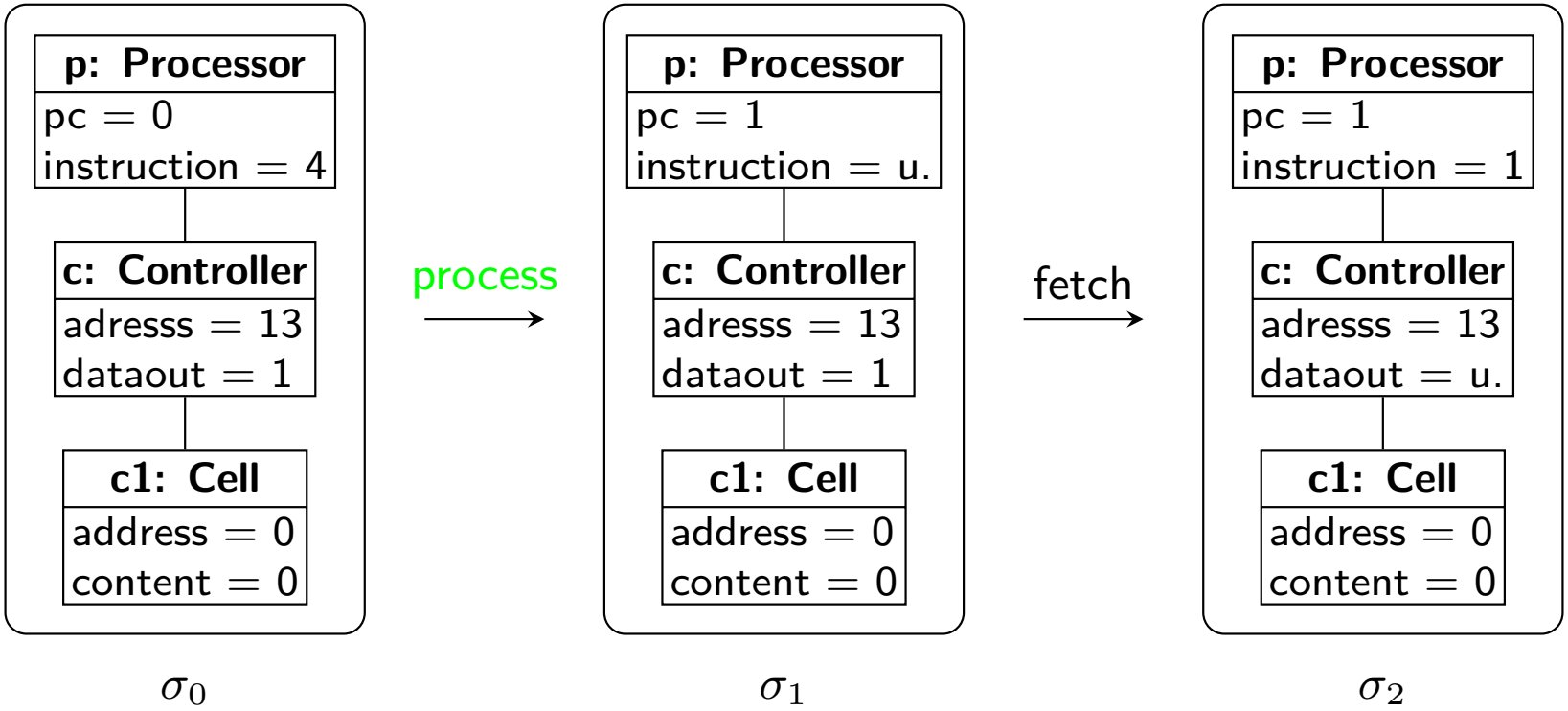
Input: UML class diagram m with OCL constraints,
operation call sequences S

1. **Operation call coverage:** How many operations from m have been called in S?
2. **Subexpression coverage:** How many constraints from m have evaluated to true during the execution of S?

Subexpression Coverage

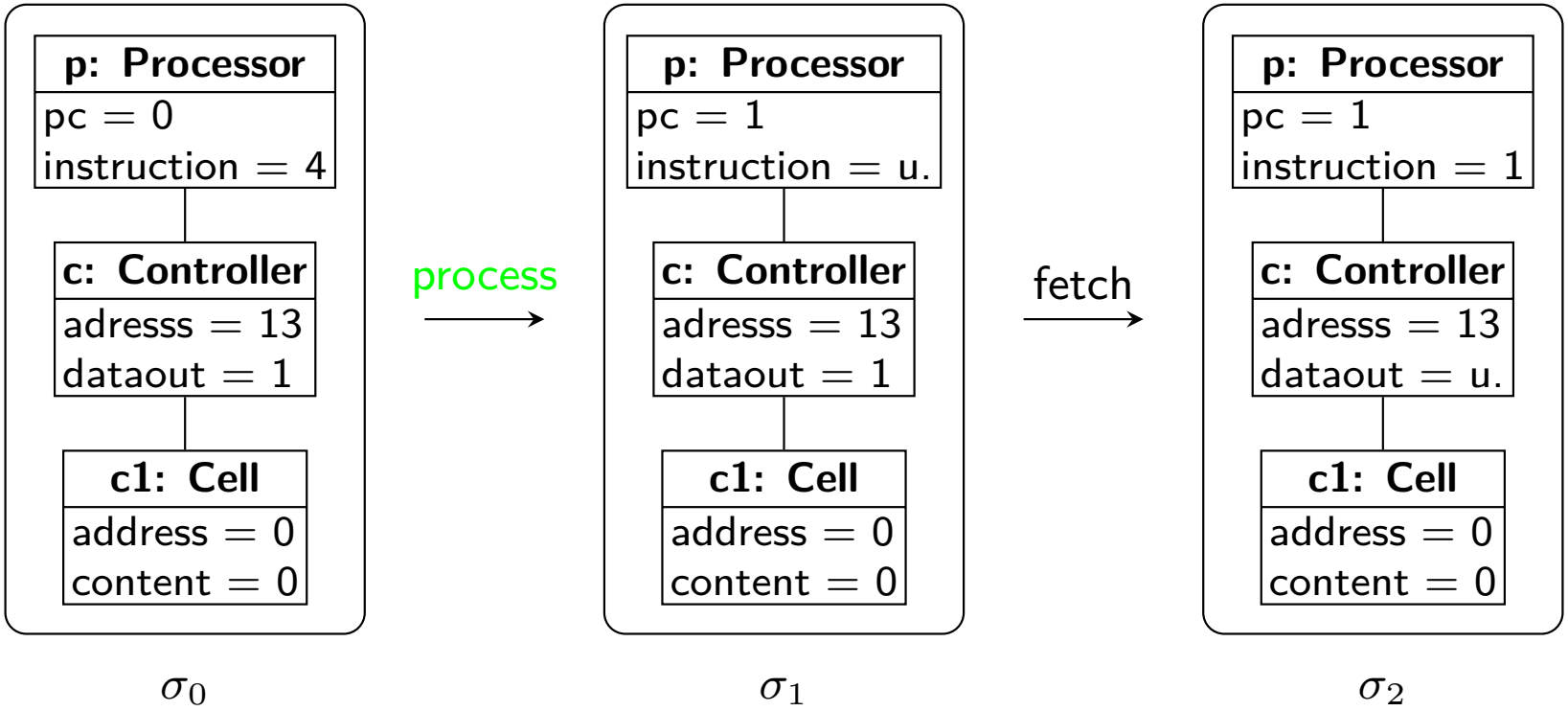


Subexpression Coverage



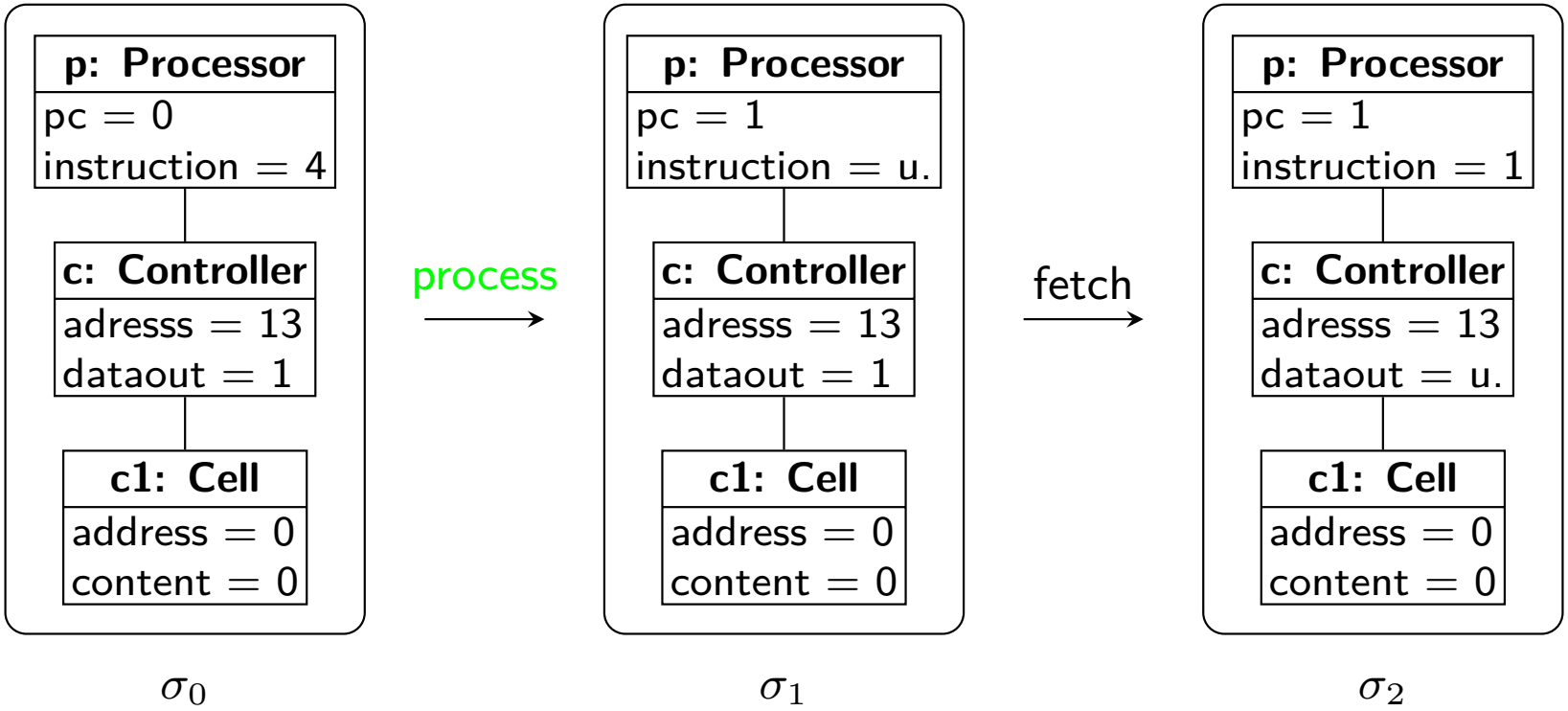
- process itself has been executed

Subexpression Coverage



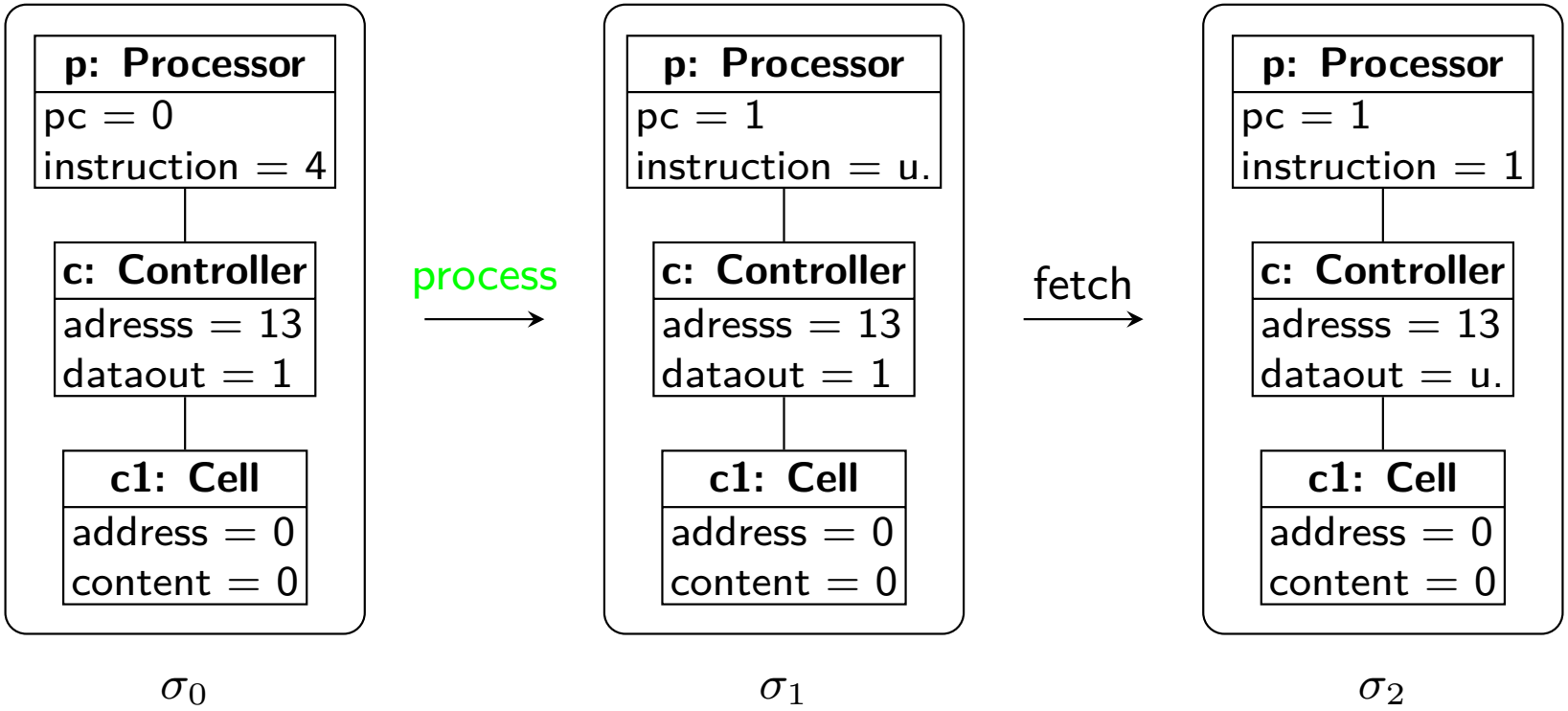
- process itself has been executed
- post24: pc@pre = 9 implies pc = 0

Subexpression Coverage



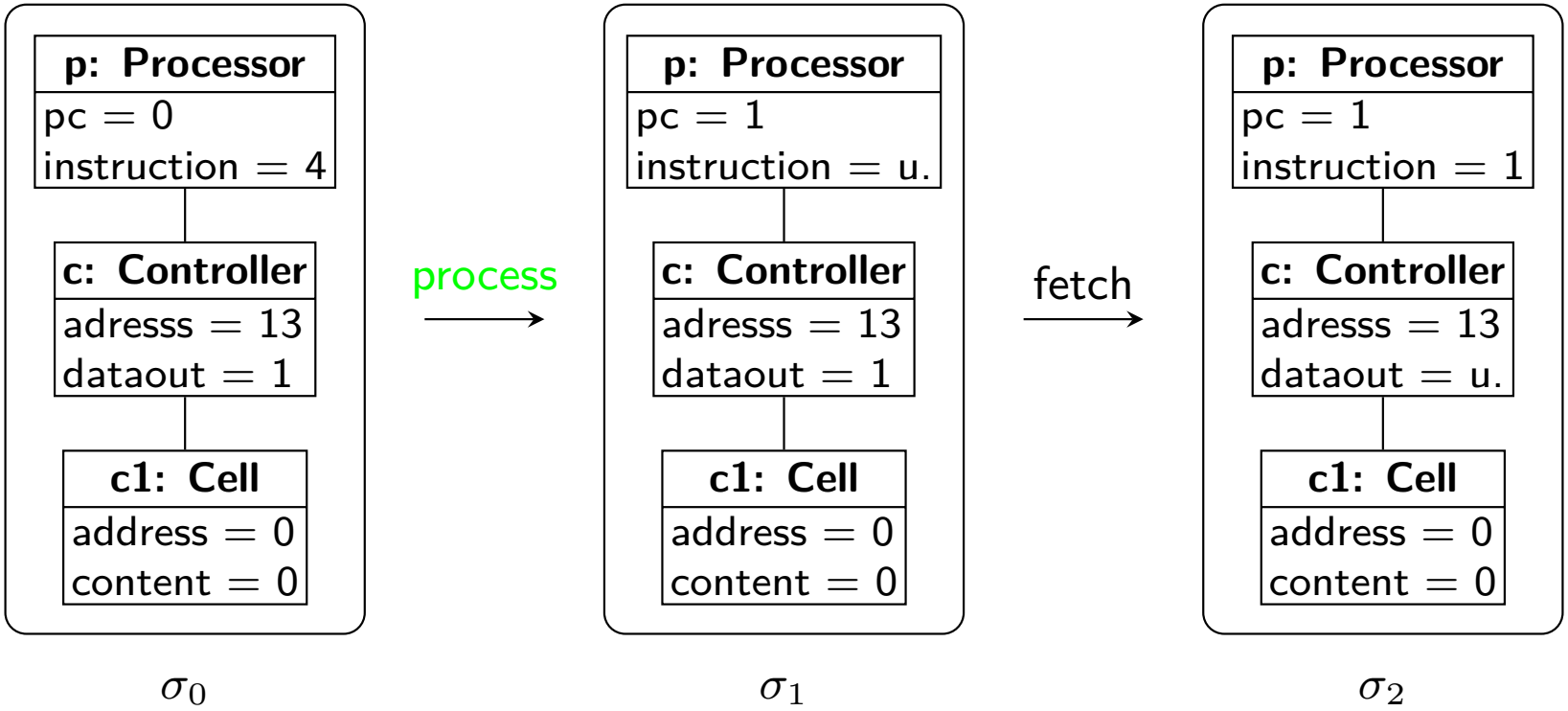
- process itself has been executed
- post24: pc@pre = 9 implies pc = 0

Subexpression Coverage



- process itself has been executed
- post24: pc@pre = 9 implies pc = 0

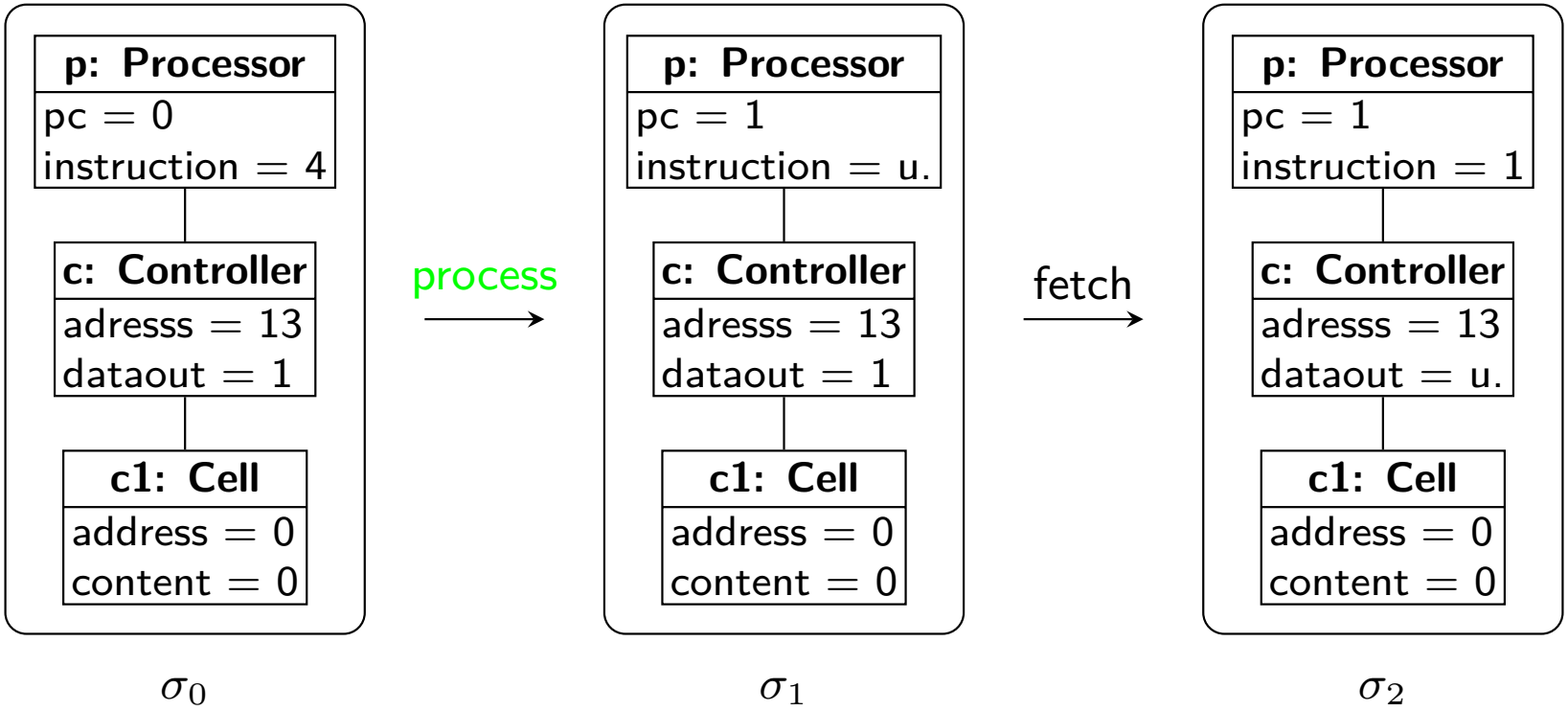
Subexpression Coverage



- process itself has been executed
- post24: pc@pre = 9 implies pc = 0

Problem: constraints might never become true

Subexpression Coverage



- process itself has been executed
- post24: pc@pre = 9 implies pc = 0

Problem: constraints might never become true → dead code

Increase Coverage

1. Calculate operation call and subexpression coverage

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased
   (a) Collect all uncovered subexpressions se

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased
   (a) Collect all uncovered subexpressions se
   (b) For each subexpression se

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased
   (a) Collect all uncovered subexpressions se
   (b) For each subexpression se
      - Add se as a pre-/postcondition to its respective operation o

Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased
   (a) Collect all uncovered subexpressions se
   (b) For each subexpression se
      - Add se as a pre-/postcondition to its respective operation o
      - Generate a sequence s' containing o

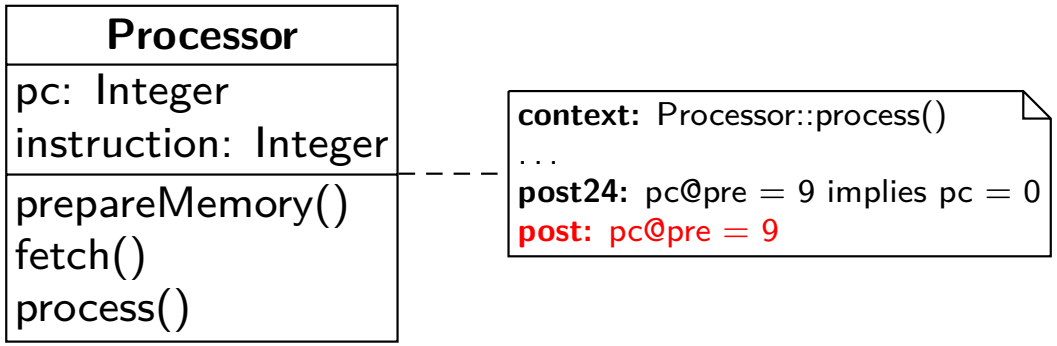
Increase Coverage

1. Calculate operation call and subexpression coverage
2. While operation call coverage $< 100\%$ OR
   Until operation call coverage cannot be further increased
   (a) Generate sequences S' containing the missing operations
   (b) Recalculate both metrics
3. While subexpression coverage $< 100\%$ OR
   Until subexpression coverage cannot be further increased
   (a) Collect all uncovered subexpressions se
   (b) For each subexpression se
      - Add se as a pre-/postcondition to its respective operation o
      - Generate a sequence s' containing o
      - Recalculate subexpression coverage

Example: Processor::process()

post24: pc@pre $= 9$ implies pc $= 0$

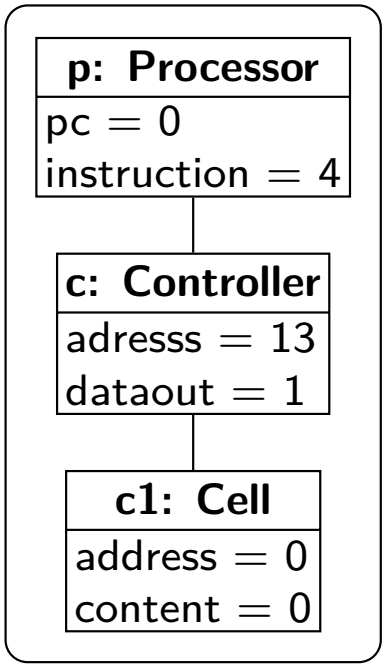
Uncovered subexpressions se: $\{$pc@pre $= 9$, pc $= 0\}$



Example: Processor::process()

post24: pc@pre $= 9$ implies pc $= 0$

Uncovered subexpressions se: $\{$pc@pre $= 9$, pc $= 0\}$

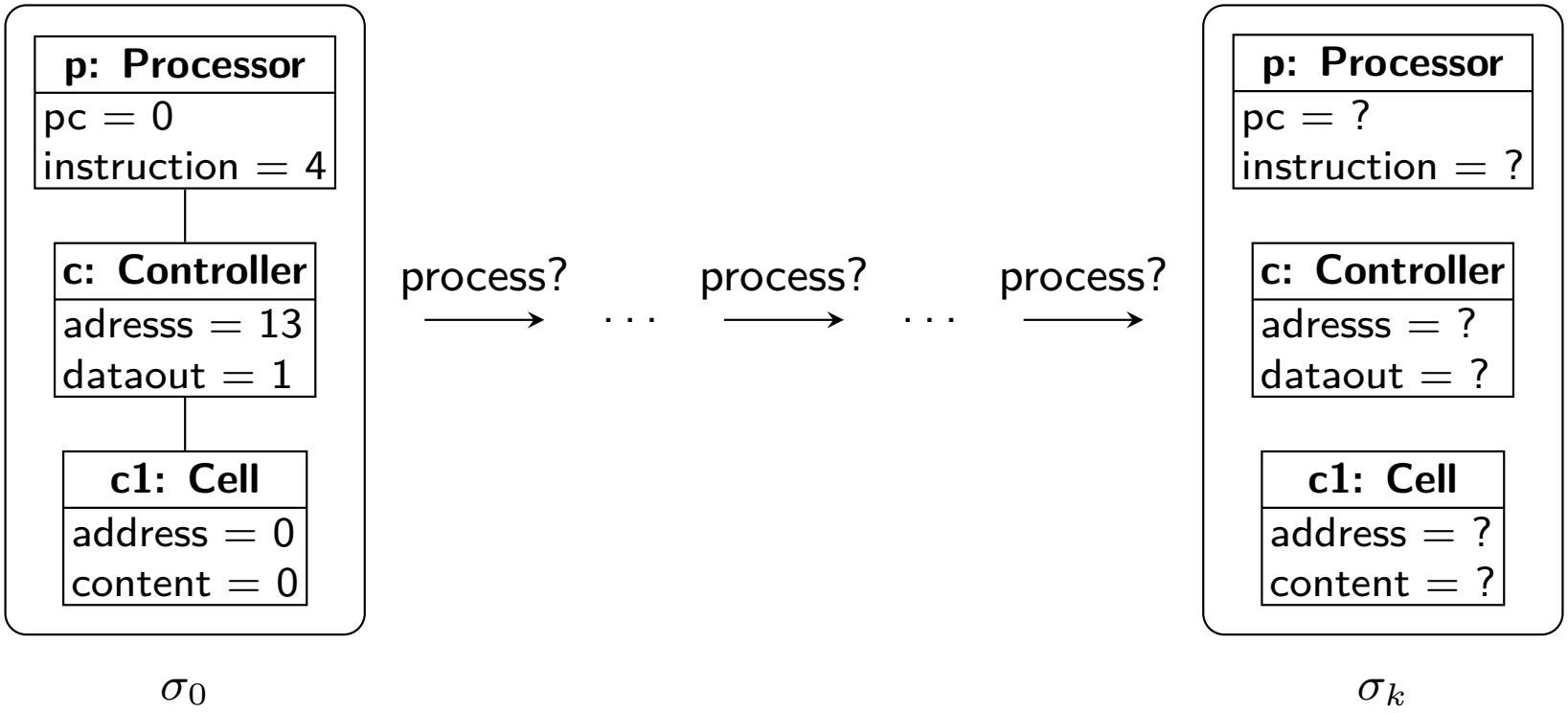


σ_0

Example: Processor::process()

post24: pc@pre $= 9$ implies pc $= 0$

Uncovered subexpressions se: $\{$pc@pre $= 9$, pc $= 0\}$



USE Plugin

Coverage

Initial Coverage | Maximum Coverage

read	Not Covered
pre3: self.address.isDefined	Not Covered
post8: self.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Not Covered
post9: self.cells->one(c : Cell | ((c.address = self.address@pre) and (c.content = self.dataout)))	Not Covered
post10: self.address.isUndefined	Not Covered
post11: (self.processor.instruction = self.processor.instruction@pre)	Not Covered
post12: (self.processor.pc = self.processor.pc@pre)	Not Covered
write	Not Covered
pre1: (self.address < 10)	Not Covered
pre2: (content < 4)	Not Covered
post1: self.cells->one(c : Cell | ((c.address = self.address) and (c.content = content)))	Not Covered
post2: self.cells->forAll(c : Cell | (c.address = c.address@pre))	Not Covered
post3: self.cells->forAll(c : Cell | ((c.address <> self.address) implies (c.content = c.content@pre)))	Not Covered
post4: (self.processor.instruction = self.processor.instruction@pre)	Not Covered
post5: (self.processor.pc = self.processor.pc@pre)	Not Covered
post6: (self.dataout = self.dataout@pre)	Not Covered
post7: (self.address = self.address@pre)	Not Covered
fetch	Not Covered
pre5: self.programMemory.dataout.isDefined	Not Covered
post18: (self.instruction = self.programMemory.dataout@pre)	Not Covered
post19: (self.pc = self.pc@pre)	Not Covered
post20: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Not Covered
post21: (self.programMemory.address = self.programMemory.address@pre)	Not Covered
post22: self.programMemory.dataout.isUndefined	Not Covered
prepareMemory	Not Covered
pre4: self.programMemory.address.isUndefined	Not Covered
post13: (self.programMemory.address = self.pc)	Not Covered
post14: (self.instruction = self.instruction@pre)	Not Covered
post15: (self.pc = self.pc@pre)	Not Covered
post16: (self.programMemory.dataout = self.programMemory.dataout@pre)	Not Covered
post17: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Not Covered
process	Covered
pre6: self.instruction.isDefined	Covered
post23: ((self.pc@pre < 9) implies (self.pc = (self.pc@pre + 1)))	Maybe Covered
post24: ((self.pc@pre = 9) implies (self.pc = 0))	Maybe Covered
post25: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Covered
post26: self.instruction.isUndefined	Covered
post27: (self.programMemory.address = self.programMemory.address@pre)	Covered
post28: (self.programMemory.dataout = self.programMemory.dataout@pre)	Covered

Initial: Covered 6/39 Elements — 15%

Maximum: Covered 14/39 Elements — 36%

USE Plugin

Coverage

Initial Coverage | Maximum Coverage

read	Not Covered
pre3: self.address.isDefined	Not Covered
post8: self.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Not Covered
post9: self.cells->one(c : Cell | ((c.address = self.address@pre) and (c.content = self.dataout)))	Not Covered
post10: self.address.isUndefined	Not Covered
post11: (self.processor.instruction = self.processor.instruction@pre)	Not Covered
post12: (self.processor.pc = self.processor.pc@pre)	Not Covered
write	Not Covered
pre1: (self.address < 10)	Not Covered
pre2: (content < 4)	Not Covered
post1: self.cells->one(c : Cell | ((c.address = self.address) and (c.content = content)))	Not Covered
post2: self.cells->forAll(c : Cell | (c.address = c.address@pre))	Not Covered
post3: self.cells->forAll(c : Cell | ((c.address <> self.address) implies (c.content = c.content@pre)))	Not Covered
post4: (self.processor.instruction = self.processor.instruction@pre)	Not Covered
post5: (self.processor.pc = self.processor.pc@pre)	Not Covered
post6: (self.dataout = self.dataout@pre)	Not Covered
post7: (self.address = self.address@pre)	Not Covered
fetch	Covered
pre5: self.programMemory.dataout.isDefined	Covered
post18: (self.instruction = self.programMemory.dataout@pre)	Covered
post19: (self.pc = self.pc@pre)	Covered
post20: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Covered
post21: (self.programMemory.address = self.programMemory.address@pre)	Covered
post22: self.programMemory.dataout.isUndefined	Covered
prepareMemory	Not Covered
pre4: self.programMemory.address.isUndefined	Not Covered
post13: (self.programMemory.address = self.pc)	Not Covered
post14: (self.instruction = self.instruction@pre)	Not Covered
post15: (self.pc = self.pc@pre)	Not Covered
post16: (self.programMemory.dataout = self.programMemory.dataout@pre)	Not Covered
post17: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Not Covered
process	Covered
pre6: self.instruction.isDefined	Covered
post23: ((self.pc@pre < 9) implies (self.pc = (self.pc@pre + 1)))	Covered
post24: ((self.pc@pre = 9) implies (self.pc = 0))	Partially Covered
post25: self.programMemory.cells->forAll(c : Cell | ((c.address = c.address@pre) and (c.content = c.content@pre)))	Covered
post26: self.instruction.isUndefined	Covered
post27: (self.programMemory.address = self.programMemory.address@pre)	Covered
post28: (self.programMemory.dataout = self.programMemory.dataout@pre)	Covered

Initial: Covered 6/39 Elements — 15%

Maximum: Covered 14/39 Elements — 36%

Experimental Evaluation

Model	Initial	Maximal	#Sequences	Run-time
CPU	0%	0%	0	<0.01s
Traffic	35%	94%	4	<0.01s
Memory	15%	97%	6	0.22s
Car	0%	100%	4	<0.01s
Life	0%	100%	3	<0.01s

Conclusions

- Two new coverage metrics for the FSL

Conclusions

- Two new coverage metrics for the FSL
- Detect dead code before the implementation

Conclusions

- Two new coverage metrics for the FSL
- Detect dead code before the implementation
- Generate operation call sequences which can be also used as test cases

Conclusions

- Two new coverage metrics for the FSL
- Detect dead code before the implementation
- Generate operation call sequences which can be also used as test cases
- Future work: more sophisticated metrics, in particular functional coverage

Coverage of OCL Operation Specifications and Invariants

Mathias Soeken, **Julia Seiter**, Rolf Drechsler

Institute of Computer Science
University of Bremen
www.informatik.uni-bremen.de/agra

July 24th 2015

@agra_uni_bremen