

Basics of Deductive Program Verification

Claude Marché

Cours MPRI 2-36-1 “Preuve de Programme”

9 décembre 2020

Preliminaries

Very first question

Lectures in English or in French?

Preliminaries

Very first question

Lectures in English or in French?

- ▶ Schedule on the Web page <https://marche.gitlabpages.inria.fr/lecture-deductive-verif/>
- ▶ Lectures 1,2,3,4: Claude Marché
- ▶ Lectures 5,6,7,8: Jean-Marie Madiot
- ▶ (to confirm) some lecture could be replaced by practical lab, support for project. Project due date around mid-February.

Preliminaries

Very first question

Lectures in English or in French?

- ▶ Schedule on the Web page <https://marche.gitlabpages.inria.fr/lecture-deductive-verif/>
- ▶ Lectures 1,2,3,4: Claude Marché
- ▶ Lectures 5,6,7,8: Jean-Marie Madiot
- ▶ (to confirm) some lecture could be replaced by practical lab, support for project. Project due date around mid-February.
- ▶ Evaluation:
 - ▶ project P using the Why3 tool (<http://why3.lri.fr>)
 - ▶ final exam E : date to decide
 - ▶ final mark = $(2E + P + \max(E, P))/4$
- ▶ internships (*stages*)

Outline

Introduction, Short History

Preliminary on Automated Deduction

- Classical Propositional Logic

- First-order logic

- Logic Theories

- Limitations of Automatic Provers

Introduction to Deductive Verification

- Formal contracts

- Hoare Logic

- Dijkstra's Weakest Preconditions

Exercises

General Objectives

Ultimate Goal

Verify that software is free of bugs

Famous software failures:

<http://www.cs.tau.ac.il/~nachumd/horror.html>

This lecture

Computer-assisted approaches for verifying that
a software conforms to a specification

Some general approaches to Verification

Static analysis, Algorithmic Verification

- ▶ *model checking* (automata-based models)
- ▶ *abstract interpretation* (domain-specific model, e.g. numerical)

Deductive verification

- ▶ formal models using expressive logics
- ▶ verification = computer-assisted mathematical proof

Some general approaches to Verification

Refinement

- ▶ Formal models
- ▶ Code derived from model, correct by construction

A long time before success

Computer-assisted verification is an old idea

- ▶ Turing, 1948
- ▶ Floyd-Hoare logic, 1969

Success in practice: only from the mid-1990s

- ▶ Importance of the *increase of performance of computers*

A first success story:

- ▶ Paris metro line 14, using *Atelier B* (1998, refinement approach)

Other Famous Success Stories

- ▶ **Flight control software of A380**: *Astree* verifies absence of run-time errors (2005, abstract interpretation)
<http://www.astree.ens.fr/>
- ▶ **Microsoft's hypervisor**: using Microsoft's *VCC* and the *Z3* automated prover (2008, deductive verification)
<http://research.microsoft.com/en-us/projects/vcc/>
More recently: verification of PikeOS
- ▶ **Certified C compiler**, developed using the *Coq* proof assistant (2009, correct-by-construction code generated by a proof assistant)
<http://compcert.inria.fr/>
- ▶ **L4.verified micro-kernel**, using tools on top of *Isabelle/HOL* proof assistant (2010, Haskell prototype, C code, proof assistant)
<http://www.ertos.nicta.com.au/research/l4.verified/>

Other Success Stories at Industry

- ▶ Frama-C
 - ▶ EDF: abstract interpretation
 - ▶ Airbus: deductive verification
- ▶ Spark/Ada: Verification of Ada programs
<https://www.adacore.com/industries>

Remark

The two above use Why3 internally

Outline

Introduction, Short History

Preliminary on Automated Deduction

- Classical Propositional Logic

- First-order logic

- Logic Theories

- Limitations of Automatic Provers

Introduction to Deductive Verification

- Formal contracts

- Hoare Logic

- Dijkstra's Weakest Preconditions

Exercises

Proposition logic in a nutshell

► Syntax:

$$\begin{aligned} \varphi ::= & \perp \mid \top \mid \mathbf{A}, \mathbf{B} && \text{(atoms)} \\ & \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg \varphi \\ & \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \end{aligned}$$

► Semantics, models: truth tables

ϕ is satisfiable if it has a model

ϕ is valid if true in all models

(equivalently $\neg\phi$ is not satisfiable)

SAT is *decidable* \rightsquigarrow SAT solvers

Demo with Why3

```
$ why3 ide propositional.mlw
```

Notice that Why3 indeed queries solvers for satisfiability of $\neg\phi$

Focus on the “Tools” menu of Why3

The screenshot displays the Why3 IDE interface. The main window shows a proof file named 'propositional.mlw' with a list of tasks (1-4) and their corresponding logical formulas. The 'Tools' menu is open, listing various solvers and proof management options. The background text in the editor includes:

```
1
2
3 (** {! MPRI lecture 2-36-1 "Proof of Programs"} *)
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The Tools menu options are:

- Alt-Ergo 2.3.1
- Coq 8.11.0
- CVC4 1.7
- Eprover 2.0
- Z3 4.8.6
- Auto level 0
- Auto level 1
- Auto level 2
- Auto level 3
- Split VC
- Edit
- Get Counterexamples
- Replay valid obsolete proofs
- Replay all obsolete proofs
- Clean node
- Remove node
- Interrupt

The background text in the editor includes:

```
1
2
3 (** {! MPRI lecture 2-36-1 "Proof of Programs"} *)
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

First-order logic in a nutshell

- ▶ Syntax:

φ	::=	...	
		$P(t, \dots, t)$	(predicates)
		$\forall x. \phi \mid \exists x. \phi$	
t	::=	x	variables
		$f(t, \dots, t)$	(function symbols)

- ▶ Semantics: models must interpret variables. C
- ▶ Satisfiability *undecidable*, but still *semi-decidable*: there exists complete systems of deduction rules (sequent calculus, natural deduction, superposition calculus)
- ▶ Examples of solvers: E, Spass, Vampire
 - Implement *refutationally complete* procedure:
if they answer 'unsat' then formula is unsatisfiable

Demo with Why3

`first-order.mlw`

Notice that Why3 logic is *typed*, and application is curried

Logic Theories

- ▶ *Theory* = set of formulas (called *theorems*) closed by logical consequence
- ▶ *Axiomatic Theory* = set of formulas generated by axioms (or axiom schemas)
- ▶ *Consistent Theory*
 - for no P , P and $\neg P$ are both theorems
 - equivalently: 'false' is not a theorem
 - equivalently: the theory has models
- ▶ *Consistent Axiomatization*
 - 'false' is not derivable

Theory of Equality

$$\forall x. x = x$$

$$\forall x, y. x = y \rightarrow y = x$$

$$\forall x, y, z. x = y \wedge y = z \rightarrow x = z$$

(congruence) for all function symbols f of arity k :

$$\forall x_1, y_1, \dots, x_k, y_k. x_1 = y_1 \wedge \dots \wedge x_k = y_k \rightarrow \\ f(x_1, \dots, x_k) = f(y_1, \dots, y_k)$$

and for all predicates p of arity k :

$$\forall x_1, y_1, \dots, x_k, y_k. x_1 = y_1 \wedge \dots \wedge x_k = y_k \rightarrow \\ p(x_1, \dots, x_k) \rightarrow p(y_1, \dots, y_k)$$

Theory of Equality, Continued

$$\forall x. x = x$$

$$\forall x, y. x = y \rightarrow y = x$$

$$\forall x, y, z. x = y \wedge y = z \rightarrow x = z$$

(congruence) ...

- ▶ General first-order deduction bad in such a case \rightsquigarrow dedicated methods
 - ▶ paramodulation
 - ▶ congruence closure (for quantifier-free fragment)
- ▶ SMT solvers (Alt-Ergo, CVC4, Z3) implement dedicated (semi-)decision procedures

Demo with Why3

`equality.mlw`

Theories Continued

Theory of a given model

= formulas true in this model

- ▶ Central example: theory of linear integer arithmetic, i.e. formulas using $+$ and \leq
 - ▶ First-order theory is known to be decidable (Presburger)
 - ▶ SMT solvers typically implement a procedure for the existential fragment
- ▶ Also: theory of (non-linear) real arithmetic is decidable (Tarski)

Non-linear Integer Arithmetic

(a.k.a. Peano Arithmetic)

First-Order Integer Arithmetic

All valid first-order formulas on integers with $+$, \times and \leq

- ▶ This theory is not even semi-decidable
- ▶ SMT solvers implement incomplete satisfiability checks:
if solver answers 'unsat' then it is unsatisfiable

Demo with Why3

`arith.mlw`

Digression about Non-linear Integer Arithmetic

Representation Theorem (Gödel)

Every recursive function f is representable by a predicate ϕ_f such that

$$\phi_f(x_1, \dots, x_k, y)$$

is true if and only if

$$y = f(x_1, \dots, x_k)$$

First incompleteness Theorem (Gödel)

That theory is not recursively axiomatizable

Summary of prover limitations

- ▶ Superposition solvers (E, Spass,)
 - ▶ do not support well theories except equality
 - ▶ do quite well with quantifiers
- ▶ SMT solvers (Alt-Ergo, CVC4, Z3)
 - ▶ several theories are built-in
 - ▶ weaker with quantifiers
- ▶ None support reasoning by induction

Outline

Introduction, Short History

Preliminary on Automated Deduction

Classical Propositional Logic

First-order logic

Logic Theories

Limitations of Automatic Provers

Introduction to Deductive Verification

Formal contracts

Hoare Logic

Dijkstra's Weakest Preconditions

Exercises

IMP language

IMP language

A very basic imperative programming language

- ▶ only global variables
- ▶ only integer values for expressions
- ▶ basic statements:
 - ▶ assignment $x \leftarrow e$
 - ▶ sequence $S_1; S_2$
 - ▶ conditionals $\text{if } e \text{ then } S_1 \text{ else } S_2$
 - ▶ loops $\text{while } e \text{ do } S$
 - ▶ no-op skip

Formal Contracts

General form of a program:

Contract

- ▶ *precondition*: expresses what is assumed before running the program
- ▶ *post-condition*: expresses what is supposed to hold when program exits

Demo with Why3

`contracts.mlw`

Hoare triples

- ▶ *Hoare triple* : notation $\{P\}s\{Q\}$
- ▶ P : formula called the *precondition*
- ▶ Q : formula called the *postcondition*

Intended meaning

$\{P\}s\{Q\}$ is true if and only if:
when the program s is executed in any state satisfying P , then
(if execution terminates) its resulting state satisfies Q

This is a *Partial Correctness*: we say nothing if s does not terminate

Examples

Examples of valid triples for partial correctness:

- ▶ $\{x = 1\}x \leftarrow x + 2\{x = 3\}$
- ▶ $\{x = y\}x \leftarrow x + y\{x = 2 * y\}$
- ▶ $\{\exists v. x = 4 * v\}x \leftarrow x + 42\{\exists w. x = 2 * w\}$
- ▶ $\{true\}while\ 1\ do\ skip\{false\}$

Running Example

Three global variables `n`, `count`, and `sum`

```
count <- 0; sum <- 1;
```

```
while sum ≤ n do
```

```
  count <- count + 1; sum <- sum + 2 * count + 1
```

Running Example

Three global variables n , $count$, and sum

```
count <- 0; sum <- 1;
```

```
while sum  $\leq$  n do
```

```
  count <- count + 1; sum <- sum + 2 * count + 1
```

What does this program compute?

(assuming input is n and output is $count$)

Running Example

Three global variables `n`, `count`, and `sum`

```
count <- 0; sum <- 1;
```

```
while sum ≤ n do
```

```
  count <- count + 1; sum <- sum + 2 * count + 1
```

What does this program compute?

(assuming input is `n` and output is `count`)

Informal specification:

- ▶ at the end of execution of this program, `count` contains the square root of `n`, rounded downward
- ▶ e.g. for `n=42`, the final value of `count` is 6.

See file `imp_isqrt.mlw`

Hoare logic as an Axiomatic Semantics

Original Hoare logic [\sim 1970]

Axiomatic Semantics of programs

Set of *inference rules* producing triples

$$\overline{\{P\}\text{skip}\{P\}}$$

$$\overline{\{P[x \leftarrow e]\}x \leftarrow e\{P\}}$$

$$\frac{\{P\}s_1\{Q\} \quad \{Q\}s_2\{R\}}{\{P\}s_1; s_2\{R\}}$$

- ▶ Notation $P[x \leftarrow e]$: replace all occurrences of program variable x by e in P .

Hoare Logic, continued

Frame rule:

$$\frac{\{P\}s\{Q\}}{\{P \wedge R\}s\{Q \wedge R\}}$$

with R a formula where no variables assigned in s occur

Consequence rule:

$$\frac{\{P'\}s\{Q'\} \quad \models P \rightarrow P' \quad \models Q' \rightarrow Q}{\{P\}s\{Q\}}$$

- ▶ Example: proof of

$$\{x = 1\}x \leftarrow x + 2\{x = 3\}$$

Hoare Logic, continued

Rules for if and while :

$$\frac{\{P \wedge e\}s_1\{Q\} \quad \{P \wedge \neg e\}s_2\{Q\}}{\{P\}\text{if } e \text{ then } s_1 \text{ else } s_2\{Q\}}$$

$$\frac{\{I \wedge e\}s\{I\}}{\{I\}\text{while } e \text{ do } s\{I \wedge \neg e\}}$$

I is called a *loop invariant*.

Example: isqrt(42)

Exercise: prove of the triple

$$\{n \geq 0\} \text{ ISQRT } \{count^2 \leq n \wedge n < (count + 1)^2\}$$

Example: isqrt(42)

Exercise: prove of the triple

$$\{n \geq 0\} \text{ ISQRT } \{count^2 \leq n \wedge n < (count + 1)^2\}$$

Could we do that by hand?

Example: `isqrt(42)`

Exercise: prove of the triple

$$\{n \geq 0\} \text{ ISQRT } \{count^2 \leq n \wedge n < (count + 1)^2\}$$

Could we do that by hand?

Back to demo: file `imp_isqrt.mlw`

Example: `isqrt(42)`

Exercise: prove of the triple

$$\{n \geq 0\} \text{ ISQRT } \{count^2 \leq n \wedge n < (count + 1)^2\}$$

Could we do that by hand?

Back to demo: file `imp_isqrt.mlw`

Warning

Finding an adequate loop invariant is a major difficulty

Beyond Axiomatic Semantics

- ▶ Operational Semantics

Beyond Axiomatic Semantics

- ▶ Operational Semantics
- ▶ Semantic Validity of Hoare Triples

Beyond Axiomatic Semantics

- ▶ Operational Semantics
- ▶ Semantic Validity of Hoare Triples
- ▶ Hoare logic as correct deduction rules

Operational semantics

[Plotkin 1981, structural operational semantics (SOS)]

- ▶ we use a standard *small-step semantics*
- ▶ *program state*: describes content of global variables at a given time. It is a finite map Σ associating to each variable x its current value denoted $\Sigma(x)$.
- ▶ Value of an expression e in some state Σ :
 - ▶ denoted $\llbracket e \rrbracket_{\Sigma}$
 - ▶ always defined, by the following recursive equations:

$$\begin{aligned}\llbracket n \rrbracket_{\Sigma} &= n \\ \llbracket x \rrbracket_{\Sigma} &= \Sigma(x) \\ \llbracket e_1 \text{ op } e_2 \rrbracket_{\Sigma} &= \llbracket e_1 \rrbracket_{\Sigma} \llbracket \text{op} \rrbracket \llbracket e_2 \rrbracket_{\Sigma}\end{aligned}$$

- ▶ $\llbracket \text{op} \rrbracket$ natural semantic of operator op on integers (with relational operators returning 0 for false and $\neq 0$ for true).

Semantics of statements

Semantics of statements: defined by judgment

$$\Sigma, s \rightsquigarrow \Sigma', s'$$

meaning: in state Σ , executing one step of statement s leads to the state Σ' and the remaining statement to execute is s' .

The semantics is defined by the following rules.

$$\frac{}{\Sigma, x \leftarrow e \rightsquigarrow \Sigma \{x \leftarrow \llbracket e \rrbracket_{\Sigma}\}, \text{skip}}$$

$$\frac{\Sigma, s_1 \rightsquigarrow \Sigma', s'_1}{\Sigma, (s_1; s_2) \rightsquigarrow \Sigma', (s'_1; s_2)}$$

$$\frac{}{\Sigma, (\text{skip}; s) \rightsquigarrow \Sigma, s}$$

Semantics of statements, continued

$$\frac{\llbracket e \rrbracket_{\Sigma} \neq 0}{\Sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \Sigma, s_1}$$

$$\frac{\llbracket e \rrbracket_{\Sigma} = 0}{\Sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \Sigma, s_2}$$

$$\frac{\llbracket e \rrbracket_{\Sigma} \neq 0}{\Sigma, \text{while } e \text{ do } s \rightsquigarrow \Sigma, (s; \text{while } e \text{ do } s)}$$

$$\frac{\llbracket e \rrbracket_{\Sigma} = 0}{\Sigma, \text{while } e \text{ do } s \rightsquigarrow \Sigma, \text{skip}}$$

Execution of programs

- ▶ \rightsquigarrow : a binary relation over pairs (state,statement)
- ▶ transitive closure : \rightsquigarrow^+
- ▶ reflexive-transitive closure : \rightsquigarrow^*

In other words:

$$\Sigma, s \rightsquigarrow^* \Sigma', s'$$

means that statement s , in state Σ , reaches state Σ' with remaining statement s' after executing some finite number of steps.

Running example:

$$\{n = 42, count = ?, sum = ?\}, ISQRT \rightsquigarrow^* \\ \{n = 42, count = 6, sum = 49\}, skip$$

Execution and termination

- ▶ any statement except `skip` can execute in any state
- ▶ the statement `skip` alone represents the final step of execution of a program
- ▶ there is no possible *runtime error*.

Definition

Execution of statement `s` in state Σ *terminates* if there is a state Σ' such that $\Sigma, s \rightsquigarrow^* \Sigma', \text{skip}$

- ▶ since there are no possible runtime errors, `s` does not terminate means that `s` *diverges* (i.e. executes infinitely).

Semantics of formulas

$\llbracket p \rrbracket_\Sigma$:

- ▶ semantics of formula p in program state Σ
- ▶ is a logic formula where no program variables appear anymore
- ▶ defined recursively as follows.

$$\begin{aligned} \llbracket e \rrbracket_\Sigma &= \llbracket e \rrbracket_\Sigma \neq 0 \\ \llbracket p_1 \wedge p_2 \rrbracket_\Sigma &= \llbracket p_1 \rrbracket_\Sigma \wedge \llbracket p_2 \rrbracket_\Sigma \\ &\vdots \end{aligned}$$

where semantics of expressions is augmented with

$$\begin{aligned} \llbracket v \rrbracket_\Sigma &= v \\ \llbracket x \rrbracket_\Sigma &= \Sigma(x) \end{aligned}$$

Notations:

- ▶ $\Sigma \models p$: the formula $\llbracket p \rrbracket_\Sigma$ is *valid*
- ▶ $\models p$: formula $\llbracket p \rrbracket_\Sigma$ holds in all states Σ .

Semantics of formulas

Other presentation of the semantics: $\llbracket p \rrbracket_{\Sigma}$:

- ▶ inline semantic of first-order formula
- ▶ $\llbracket e \rrbracket_{\Sigma, \nu}$ with ν mapping of logic variables to integers.
- ▶ defined recursively as follows.

$$\begin{aligned}\llbracket p_1 \wedge p_2 \rrbracket_{\Sigma, \nu} &= \begin{cases} \top & \text{if } \llbracket p_1 \rrbracket_{\Sigma, \nu} = \top \text{ and } \llbracket p_2 \rrbracket_{\Sigma, \nu} = \top \\ \perp & \end{cases} \\ \llbracket \forall x. e \rrbracket_{\Sigma, \nu} &= \top \text{ if for all } \nu'. \llbracket e \rrbracket_{\Sigma, \nu[x \leftarrow \nu']} = \top \\ &\vdots\end{aligned}$$

where semantics of expressions is augmented with

$$\begin{aligned}\llbracket v \rrbracket_{\Sigma, \nu} &= \nu(v) \\ \llbracket x \rrbracket_{\Sigma, \nu} &= \Sigma(x)\end{aligned}$$

Soundness

Definition (Partial correctness)

Hoare triple $\{P\}s\{Q\}$ is said *valid* if:

for any states Σ, Σ' , if

- ▶ $\Sigma, s \rightsquigarrow^* \Sigma', \text{skip}$ and
- ▶ $\Sigma \models P$

then $\Sigma' \models Q$

Theorem (Soundness of Hoare logic)

The set of rules is correct: any derivable triple is valid.

This is *proved by induction on the derivation tree* of the considered triple.

For each rule: assuming that the triples in premises are valid, we show that the triple in conclusion is valid too.

Annotated Programs

Goal

Add automation to the Hoare logic approach

We augment IMP with *explicit loop invariants*

```
while  $e$  invariant / do  $s$ 
```

Weakest liberal precondition

[Dijkstra 1975]

Function $WLP(s, Q)$:

- ▶ s is a statement
- ▶ Q is a formula
- ▶ returns a formula

It should return the *minimal precondition* P that validates the triple $\{P\}s\{Q\}$

Definition of $WLP(s, Q)$

Recursive definition:

$$\begin{aligned}WLP(\text{skip}, Q) &= Q \\WLP(x \leftarrow e, Q) &= Q[x \leftarrow e] \\WLP(s_1; s_2, Q) &= WLP(s_1, WLP(s_2, Q)) \\WLP(\text{if } e \text{ then } s_1 \text{ else } s_2, Q) &= \\& (e \rightarrow WLP(s_1, Q)) \wedge (\neg e \rightarrow WLP(s_2, Q))\end{aligned}$$

Definition of $WLP(s, Q)$, continued

$$\begin{aligned} WLP(\text{while } e \text{ invariant } I \text{ do } s, Q) = & \\ & I \wedge \quad \text{(invariant true initially)} \\ & \forall v_1, \dots, v_k. \\ & \quad ((e \wedge I) \rightarrow WLP(s, I)) \quad \text{(invariant preserved)} \\ & \quad \wedge ((\neg e \wedge I) \rightarrow Q)[w_i \leftarrow v_i] \quad \text{(invariant implies post)} \end{aligned}$$

where w_1, \dots, w_k is the set of assigned variables in statement s and v_1, \dots, v_k are fresh logic variables

Examples

$$\text{WLP}(x < x + y, x = 2y) \equiv x + y = 2y$$

Examples

$$\text{WLP}(x \leftarrow x + y, x = 2y) \equiv x + y = 2y$$

$$\text{WLP}(\text{while } y > 0 \text{ invariant } \textit{even}(y) \text{ do } y \leftarrow y - 2, \textit{even}(y)) \equiv$$

Examples

$$\text{WLP}(x \leftarrow x + y, x = 2y) \equiv x + y = 2y$$

$$\begin{aligned} \text{WLP}(\text{while } y > 0 \text{ invariant } \textit{even}(y) \text{ do } y \leftarrow y - 2, \textit{even}(y)) &\equiv \\ \textit{even}(y) \wedge & \\ \forall v, ((v > 0 \wedge \textit{even}(v)) \rightarrow \textit{even}(v - 2)) & \\ \wedge ((v \leq 0 \wedge \textit{even}(v)) \rightarrow \textit{even}(v)) & \end{aligned}$$

Soundness

Theorem (Soundness)

For all statement s and formula Q , $\{WLP(s, Q)\}s\{Q\}$ is valid.

Proof by induction on the structure of statement s .

Consequence

For proving that a triple $\{P\}s\{Q\}$ is valid, it suffices to prove the formula $P \rightarrow WLP(s, Q)$.

This is basically the goal that Why3 produces

Digression: Completeness of Hoare Logic

Two major difficulties for proving a program

- ▶ *guess the appropriate intermediate formulas* (for sequence, for the loop invariant)
- ▶ *prove the logical premises of consequence rule*

Digression: Completeness of Hoare Logic

Two major difficulties for proving a program

- ▶ *guess the appropriate intermediate formulas* (for sequence, for the loop invariant)
- ▶ *prove the logical premises of consequence rule*

Theoretical question: completeness. Are all valid triples derivable from the rules?

Theorem (Relative Completeness of Hoare logic)

*The set of rules of Hoare logic is **relatively** complete: if the logic language is **expressive enough**, then any valid triple $\{P\}s\{Q\}$ can be derived using the rules.*

Digression: Completeness of Hoare Logic

Two major difficulties for proving a program

- ▶ *guess the appropriate intermediate formulas* (for sequence, for the loop invariant)
- ▶ *prove the logical premises of consequence rule*

Theoretical question: completeness. Are all valid triples derivable from the rules?

Theorem (Relative Completeness of Hoare logic)

The set of rules of Hoare logic is relatively complete: if the logic language is expressive enough, then any valid triple $\{P\}s\{Q\}$ can be derived using the rules.

[Cook, 1978] “Expressive enough”: representability of any recursive function

Yet, this does not provide an effective recipe to discover suitable loop invariants (see also the theory of abstract interpretation *[Cousot, 1990]*)

Outline

Introduction, Short History

Preliminary on Automated Deduction

Classical Propositional Logic

First-order logic

Logic Theories

Limitations of Automatic Provers

Introduction to Deductive Verification

Formal contracts

Hoare Logic

Dijkstra's Weakest Preconditions

Exercises

Exercise 1

Consider the following (inefficient) program for computing the sum $a + b$.

```
x <- a; y <- b;  
while y > 0 do  
  x <- x + 1; y <- y - 1
```

(Why3 file to fill in: `imp_sum.mlw`)

- ▶ Propose a post-condition stating that the final value of x is the sum of the values of a and b
- ▶ Find an appropriate loop invariant
- ▶ Prove the program.

Exercise 2

The following program is one of the original examples of Floyd.

```
q <- 0; r <- x;
while r ≥ y do
  r <- r - y; q <- q + 1
```

(Why3 file to fill in: `imp_euclide.mlw`)

- ▶ Propose a formal precondition to express that x is assumed non-negative, y is assumed positive, and a formal post-condition expressing that q and r are respectively the quotient and the remainder of the Euclidean division of x by y .
- ▶ Find appropriate loop invariant and prove the correctness of the program.

Exercise 3

Let's assume given in the underlying logic the functions $\text{div2}(x)$ and $\text{mod2}(x)$ which respectively return the division of x by 2 and its remainder. The following program is supposed to compute, in variable r , the power x^n .

```
r ≤ 1; p <- x; e <- n;  
while e > 0 do  
  if mod2(e) ≠ 0 then r <- r * p;  
  p <- p * p;  
  e <- div2(e);
```

(Why3 file to fill in: `power_int.mlw`)

- ▶ Assuming that the power function exists in the logic, specify appropriate pre- and post-conditions for this program.
- ▶ Find an appropriate loop invariant, and prove the program.

Exercise 4

The Fibonacci sequence is defined recursively by $fib(0) = 0$, $fib(1) = 1$ and $fib(n + 2) = fib(n + 1) + fib(n)$. The following program is supposed to compute fib in linear time, the result being stored in y .

```
y <- 0; x <- 1; i <- 0;
while i < n do
  aux <- y; y <- x; x <- x + aux; i <- i + 1
```

- ▶ Assuming fib exists in the logic, specify appropriate pre- and post-conditions.
- ▶ Prove the program.

Exercise (Exam 2011-2012)

In this exercise, we consider the simple language of the first lecture of this course, where expressions do not have any side effect.

1. *Prove that the triple*

$$\{P\}x \leftarrow e\{\exists v, e[x \leftarrow v] = x \wedge P[x \leftarrow v]\}$$

is valid with respect to the operational semantics.

2. *Show that the triple above can be proved using the rules of Hoare logic.*

Let us assume that we replace the standard Hoare rule for assignment by the rule

$$\overline{\{P\}x \leftarrow e\{\exists v, e[x \leftarrow v] = x \wedge P[x \leftarrow v]\}}$$

3. *Show that the triple $\{P[x \leftarrow e]\}x \leftarrow e\{P\}$ can be proved with the new set of rules.*

Bibliography

- Cook(1978) S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978. doi: 10.1137/0207005.
- Cousot(1990) P. Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 841–993. North-Holland, 1990.
- Dijkstra(1975) E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975. ISSN 0001-0782. doi: 10.1145/360933.360975.

Bibliography

- Floyd(1967) R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, Providence, Rhode Island, 1967. American Mathematical Society.
- Hoare(1969) C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580 and 583, Oct. 1969.
- Plotkin(2004) G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:3–15, 2004. doi: 10.1016/j.jlap.2004.03.009.