15-812: Programming Language Semantics

Lecture Notes on Proving Loops

André Platzer

Carnegie Mellon University
Lecture 6

1 Introduction

The previous lecture provided axioms for compositional reasoning about deterministic sequential programs. All the axioms compositionally reduce the truth of a postcondition of a more complex program to a logical combination of postconditions of simpler programs. All axioms? Well, all the axioms but one: those about loops.

But putting loops aside for the moment, these axioms completely tell us what we need to do to understand a program. All we need to do is to identify the top-level operator of the program and apply the corresponding axiom from its left hand side to its structurally simpler right hand side, which will eventually reduce the property of a program to first-order logic with arithmetic but without programs. This process is completely systematic.

So except for the (nontrivial) fact that we will have to hope that an SMT solver will be able to handle the remaining arithmetic, our "only" problem is what we could possibly to with a loop. The unwinding axioms from the previous lecture were only partially helpful, which is why this lecture investigates more comprehensive reasoning techniques for loops. We follow an entirely systematic approach [Pla18, Chapter 7] to understanding loop invariants, an induction technique for loops, which is of central significance for program verification. We will also experience our share of the important phenomenon of loop invariant search.

2 Recall: Loop the Loop

Recall the two (equivalent) axioms for handling while-loops by unwinding the loop:

[unwind] $[\mathsf{while}(Q)\,\alpha]P \leftrightarrow [\mathsf{if}(Q)\,\{\alpha;\mathsf{while}(Q)\,\alpha\}]P$

[unfold] $[\mathsf{while}(Q)\,\alpha]P \leftrightarrow (Q \rightarrow [\alpha][\mathsf{while}(Q)\,\alpha]P) \wedge (\neg Q \rightarrow P)$

Especially the [unfold] axiom makes it very apparent that the deficiency with both axioms is that, when used from left to right, they reduce a property of a while loop to some logic and then the same property of the same loop again. While the isolated α loop body can be handled with the other axioms, the $\mathsf{while}(Q)\,\alpha$ loop is still remaining and could be handled by another [unfold] but then the same issue persists. This principle of successively unrolling the loop is still perfectly helpful for loops that terminate right away or that always terminate after 1 rounds or after 2 rounds or after some other fixed finite maximum number of iterations such as 5. But "most" loops are not like that. If the loop terminates after a very large number of loop iterations, or if we cannot know ahead of time after which fixed natural number of loop iterations it terminates, then unrolling the loop does not help, because there will always be a conjunct referring to what happens if the loop repeats again.[1]

3 Loops

In order to resolve these issues with how to prove loops, we will follow a completely systematic approach [Pla18] to develop compositional proof principles for loops. Successive loop unrolling with the [unfold] axiom ran into the difficulty that it had to predict perfectly when the loop stops because the loop condition Q is false. The number of iterations for a while loop is hard to predict. It was of course defined exactly in the semantics:

5. $I[\![\mathsf{while}(Q)\,\alpha]\!] = \{(\omega,\nu) :$ there are an n and states $\mu_0 = \omega, \mu_1, \mu_2, \ldots, \mu_n = \nu$ such that for all $0 \leq i < n$: ① the loop condition is true $\mu_i \in I[\![Q]\!]$ and ② from state μ_i is state μ_{i+1} reachable by running α so $(\mu_i, \mu_{i+1}) \in I[\![\alpha]\!]$ and ③ the loop condition is false $\mu_n \notin I[\![Q]\!]$ in the end$\}$

But mapping this exact termination of while loops into logic will be a distraction from the essential aspects. In order to understand the principle of repetition in loops we will, instead, make it principally unpredictable when exactly the loop terminates by entirely removing the loop guard Q. For one thing, it is easier to understand the principle of repetition without simultaneously having to worry about the impact that loop guards have. After understanding the principle of repetition, we will then come back to apply the knowledge we gained from that excursion to the original question of while loops.

[1] There is a very subtle argument why such unrolling is still enough progress to prove properties of loops [Pla15], but this is beyond the scope here.

4 Nondeterministic Repetition

In order to understand the principle of repetition, we will, in this lecture, investigate the *nondeterministic repetition* α^*. The effect of the nondeterministic repetition α^* is to repeat the program α any arbitrary n number of times for any nondeterministically chosen natural number $n \in \mathbb{N}$. We cannot predict n. Just like in a regular expression such as $\texttt{a}^*$ which matches any natural number of occurrences of the letter a, for example aaa or aaaaa, the nondeterministic repetition α^* repeats α any number of times, for example $\alpha;\alpha;\alpha$ or $\alpha;\alpha;\alpha;\alpha;\alpha$. Since the nondeterministic repetition α^* can repeat program α any arbitrary number of times, this makes the resulting programs nondeterministic, because they can run in more than one way.

In fact, this is a little bit like what happens for $\mathsf{while}(Q)\,\alpha$ loops in practice, too. In principle, the computations of $\mathsf{while}(Q)\,\alpha$ are deterministic because from every initial state there is at most one run of this program and this run takes some deterministic number of loop iterations. But in practice, it's not like we could easily tell how often exactly a $\mathsf{while}(Q)\,\alpha$ loop repeats. If we could we would have solved the halting problem, which Church-Turing thought of as a difficult one. So when we try to understand a $\mathsf{while}(Q)\,\alpha$ loop, realistically, we would also often have to say that this loop might repeat any number of times, just because we don't know any better.

For the sake of better understanding while loops, let's extend the syntax as follows:[2]

Definition 1 (Nondeterministic program)**.** *Nondeterministic while programs* are defined by extending the grammar of deterministic while programs with one additional case, highlighted in **bold**:

$$\alpha, \beta \;::=\; x := e \mid ?Q \mid \mathsf{if}(Q)\,\alpha\,\mathsf{else}\,\beta \mid \alpha;\beta \mid \mathsf{while}(Q)\,\alpha \mid \boldsymbol{\alpha^*}$$

Of course, as soon as we add a new operator into our syntax, we have to give it a meaning. The meaning of nondeterministic repetition is quite different from the meaning of all the other deterministic program operators, precisely because its effect is nondeterministic. But our semantics of programs is already perfectly prepared for that, because it is a relation $I[\![\alpha]\!] \subseteq \mathcal{S} \times \mathcal{S}$ on states. In deterministic programs, at most one final state is reachable from every initial state. In nondeterministic programs, instead, it can also happen that multiple states are reachable. The nondeterministic while program $\{x := x + 2\}^*$ for example will repeatedly increment variable x by 2 for any number of times. It might increment x by 2 or by 10 or by 0 or by 414 or ..., but not by 3 because that's an odd number. Even if this may sound like a lot of options, it turns out that the semantics of nondeterministic repetition is actually much easier than that of while loops, precisely because we do not need to keep track of when exactly it exits on account of the loop guard Q.

[2] Nondeterministic programs usually include a slightly different set of operators [Pra76, HKT00]. Here we consider nondeterministic while programs which only involve a minimal change compared to deterministic while programs.

Definition 2 (Transition semantics of nondeterministic while programs)**.** Each nondeterministic while program α is interpreted semantically as a binary reachability relation $I[\![\alpha]\!] \subseteq \mathcal{S} \times \mathcal{S}$ over states, defined inductively by extending the definition for deterministic while programs with the following case

6. $I[\![\alpha^*]\!] = \{(\omega, \nu) \ : \ $ there are an n and states $\mu_0 = \omega, \mu_1, \mu_2, \ldots, \mu_n = \nu$ such that $(\mu_i, \mu_{i+1}) \in I[\![\alpha]\!]$ for all $0 \leq i < n\}$
   That is, state μ_{i+1} is reachable from state μ_i by running α for all i.

Comparing the definition of the semantics, the meaning of nondeterministic repetition is much easier to define even if it allows more behavior, because all it says is that the nondeterministic repetition α^* repeats (and leaves open how often exactly). But while loops are easy to get back from nondeterministic repetitions and tests, because while loops $\mathsf{while}(Q)\,\alpha$ are equivalent to guarding the loop body of a nondeterministic repetition by the test $?Q$ and guarding the loop exit by the test $?\neg Q$ so that no execution can succeed that stops too early or too late:

$$\mathsf{while}(Q)\,\alpha \ \equiv \ \{?Q;\alpha\}^*;?\neg Q \tag{1}$$

This equivalence of while programs with nondeterministic repetitions using suitable tests gives us confidence that we will later be able to understand while loops if we just first understand nondeterministic repetition itself.

Remember that the box modality in the formula $[\alpha^*]P$ considers *all* possible executions of the nondeterministic repetition α^*. So $[\alpha^*]P$ really says, but more concisely, that the following infinite collection of formulas is true:

$$P, \quad [\alpha]P, \quad [\alpha;\alpha]P, \quad [\alpha;\alpha;\alpha]P, \quad [\alpha;\alpha;\alpha;\alpha]P, \quad [\alpha;\alpha;\alpha;\alpha;\alpha]P, \quad [\alpha;\alpha;\alpha;\alpha;\alpha;\alpha]P, \quad \ldots$$

5 Unwinding Nondeterministic Repetitions

Of course it would be very easy to also design and justify an axiom that unwinds a nondeterministic repetition, just like axioms [unwind] and [unfold] do for while loops:

$$[^*] \quad [\alpha^*]P \leftrightarrow P \wedge [\alpha][\alpha^*]P$$

But that axiom also still shares the exact same problem of reducing a property of a nondeterministic repetition to a logical combination involving the same property of the same nondeterministic repetition.

6 Induction

There isn't much that we can do to improve matters in how the iteration axiom [*] insists on the postcondition P in the first conjunct, because nondeterministic loops are allowed to repeat 0 times, which keeps them in the initial state. So unless we show that the postcondition P is true in the initial state, the property $[\alpha^*]P$ can never be true. But

the second conjunct of axiom [*] retains the exact same property $[\alpha^*]P$ after $[\alpha]$. Let's develop a new axiom of the form:

$$[\alpha^*]P \leftrightarrow P \wedge \ldots$$

What we definitely need to show in addition to P is that $[\alpha]P$ is true. But since we already showed that P is true in the first conjunct, it is enough for us to show the implication $P \rightarrow [\alpha]P$. Unfortunately, showing just those two conditions is not enough:

$$[\alpha^*]P \leftrightarrow P \wedge (P \rightarrow [\alpha]P)$$

because the second conjunct only says that the implication $P \rightarrow [\alpha]P$ is true in the current state, which says nothing about states that are reached after the loop α^* ran repeatedly, say, for 10 times. We need to know that the implication $P \rightarrow [\alpha]P$ also holds again after the loop ran a bunch more times.

These thoughts lead to the induction axiom for loops [Pla18]:

Lemma 3. *The induction axiom I is sound:*

$$\text{I}\quad [\alpha^*]P \leftrightarrow P \wedge [\alpha^*](P \rightarrow [\alpha]P)$$

Proof. In order to prove validity, we consider any state ω and show that

$$\omega \in I[\![[\alpha^*]P \leftrightarrow P \wedge [\alpha^*](P \rightarrow [\alpha]P)]\!]$$

As usual the proof considers each direction separately.

"$\rightarrow$" This direction is easy to see because a nondeterministic repetition α^* is allowed to repeat 0 times such that $\omega \in I[\![[\alpha^*]P]\!]$ implies $\omega \in I[\![P]\!]$. Also if P is true after any number of repetitions of α^* then also $\omega \in I[\![[\alpha^*][\alpha]P]\!]$ after at least one iteration. This implies the right hand side.

"$\leftarrow$" This direction is by induction on the number n of loop iterations.

- $n = 0$: The first conjunct implies P holds in the final state, which is the initial state ω after 0 repetitions.
- $n + 1$: By induction hypothesis, P is always true after n repetitions from initial state ω. In order to show that P is also always true after $n + 1$ repetitions from ω, consider any intermediate state μ such that $(\omega, \mu) \in I[\![\alpha^*]\!]$ with n iterations and any final state ν with $(\mu, \nu) \in I[\![\alpha]\!]$. By induction hypothesis, $\mu \in I[\![P]\!]$. By the right conjunct of the assumption also $\mu \in I[\![P \rightarrow [\alpha]P]\!]$. Consequently, $\nu \in I[\![P]\!]$. □

7 Loop Proofs

The induction axiom I is a wonderful equivalence but it still comes with the challenge of reducing a property of a loop to another property of the same loop. Even if the

other property $P \to [\alpha]P$ comes with an assumption to use, it's still a property of a loop. But now we can combine the induction axiom I with another important proof principle: generalization. Gödel's generalization rule G says that one way of proving a postcondition of a box modality is to just prove the postcondition itself:

$$\text{G}\ \frac{\vdash P}{\Gamma \vdash [\alpha]P, \Delta}$$

Indeed, if P has a proof then it is valid by soundness, so true in all states, hence also true in all states after running program α. Of course, Gödel's generalization rule G cannot soundly keep any information from Γ, Δ for the premise, because it might no longer be true after α. Using the Gödel rule G after the loop induction axiom I reduces the proof of $[\alpha^*]P$ to a proof that P is true in the initial state and to a proof that the implication $P \to [\alpha]P$ is valid, so true in all states:

$$\text{ind}'\ \frac{\Gamma \vdash P, \Delta \quad P \vdash [\alpha]P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

This proof rule ind′ says that for proving $[\alpha^*]P$ from assumptions Γ with alternatives Δ (conclusion) it suffices to prove the postcondition P from assumptions Γ with alternatives Δ (left premise) in addition to proving that the postcondition P is inductive so $[\alpha]P$ is true in any state where P is true (right premise). Proving that rule ind′ is sound is easy by deriving it from axiom I and rule G.

Lemma 4. *The basic loop induction rule ind′ is a derived rule and, thus, sound:*

$$\text{ind}'\ \frac{\Gamma \vdash P, \Delta \quad P \vdash [\alpha]P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Proof. Showing that rule ind′ is a derived rule requires us to derive its conclusion in sequent calculus from its premises, which we derive from axiom I with rule G:

$$\text{I}\ \frac{\text{$\wedge$R}\ \dfrac{\Gamma \vdash P, \Delta \qquad \text{G}\ \dfrac{\text{$\to$R}\ \dfrac{P \vdash [\alpha]P}{\vdash P \to [\alpha]P}}{\Gamma \vdash [\alpha^*](P \to [\alpha]P), \Delta}}{\Gamma \vdash P \wedge [\alpha^*](P \to [\alpha]P), \Delta}}{\Gamma \vdash [\alpha^*]P, \Delta}$$

□

8 Loop Invariants

Proof rule ind′ properly reduces the proof of a nondeterministic repetition to a proof of subquestions that do not involve the repetition again. Its only downside is that the rule no longer comes in the form of an equivalence axiom. And indeed there are cases where the ind′ rule does not work like it should. How could that happen?

Before you read on, see if you can find the answer for yourself.

Everything that proof rule ind′ proves is valid, after all the rule is sound because it is derived from sound axioms and proof rules. There are, however, cases where proof rule ind′ does not prove the conclusion even though it is valid. The problem is very apparent from how rule ind′ is derived with the help of the G rule which misplaces a whole $[\alpha^*]$ modality. That might have contained valuable information about what exactly changes as the loop runs, which is lost when setting out for an isolated proof of the postcondition.

We can make up for that by retaining a little more information about the long history of loop body executions by providing a little more information in a loop invariant J that we choose freely and perform induction with invariant J instead. Of course, we then also have to prove that the loop invariant J we dreamed up implies the original postcondition we were interested in (third premise):

$$\text{loop}\ \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

This rule can easily be derived from the monotonicity principle that if P implies Q then if P is always true after running α then Q is also always true after running α:

$$\text{M}[\cdot]\ \frac{P \vdash Q}{\Gamma, [\alpha]P \vdash [\alpha]Q, \Delta}$$

Lemma 5. *Loop rule loop is a derived rule and thus sound:*

$$\text{loop}\ \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Proof. The proof rule loop can be derived from rule ind′ by rule M[·]:

$$\text{cut}\ \frac{\text{ind}'\ \dfrac{\Gamma \vdash J, \Delta \qquad J \vdash [\alpha]J}{\Gamma \vdash [\alpha^*]J, \Delta} \qquad \text{M}[\cdot]\ \dfrac{J \vdash P}{\Gamma, [\alpha^*]J \vdash [\alpha^*]P, \Delta}}{\Gamma \vdash [\alpha^*]P, \Delta}$$

□

The DL sequent calculus consists of the axioms that we have seen already oriented into the direction that turns properties of complex programs into properties of simpler programs. It also includes an assignment axiom that takes care of renaming variables appropriately. In the left most branch, rule ind proves that the induction invariant I is true in the beginning. On the middle branch, rule ind proves that the invariant is true again after executing the loop body α once, if only I was true before executing the loop body and the loop test Q was true (otherwise the loop doesn't execute). On the right branch, rule ind proves that the invariant I together with the knowledge that the loop test Q must have failed for the loop to terminate at all imply the original postcondition ϕ.

9 Loop Invariants for While Loops

This is a great answer for nondeterministic repetitions α^* that repeat α any number of times but our actual interest was in understanding the $\mathsf{while}(Q)\,\alpha$ loop which says precisely when to repeat and when to stop according to the loop guard Q. Let's take what we learned about repetition by and large from α^* and apply it back to while loops.

The following version of the loop invariant rule for $\mathsf{while}(Q)\,\alpha$ loops (which we simply call while) can be derived from the loop rule for nondeterministic repetitions α^* using the definition of the former using the latter from (1):

$$\text{while}\ \frac{\Gamma \vdash J, \Delta \quad J, Q \vdash [\alpha]J \quad J, \neg Q \vdash P}{\Gamma \vdash [\mathsf{while}(Q)\,\alpha]P, \Delta}$$

Proving that this while rule for while loops is a derived proof rule is an excellent exercise. The rule is also an excellent example how the study of something more general can provide systematic insights about something more specific.

10 Proving a Loopy Program

Enough theory. Let's turn to an actual program with a loop that we would like to prove. Consider the following program:

```
s := 0;
i := 0;
while (i<x) {
  s := s+2*i+1;
  i := i+1
}
```

What does this program do? How can we prove it?

Before you read on, see if you can find the answer for yourself.

$$[:=]_= \frac{\Gamma, y = e \vdash p(y), \Delta}{\Gamma \vdash [x := e]p(x), \Delta}\ (y \text{ new}) \qquad =\text{R}\ \frac{\Gamma, x = e \vdash p(e), \Delta}{\Gamma, x = e \vdash p(x), \Delta} \qquad =\text{L}\ \frac{\Gamma, x = e, p(e) \vdash \Delta}{\Gamma, x = e, p(x) \vdash \Delta}$$

Figure 1: Some proof rules related to equations

11 Sum Up the Square

Let β be the above while program. We set out to prove the DL formula $[\beta]s = x * x$ saying that the program β always computes the square of x in variable s in this section. For the most part, the proof of DL formula $[\beta]s = x * x$ is completely canonical. The one step that is not is, of course, also the most difficult one. The while proof rule expects a loop invariant J as input. Do you have a good idea?

Let's proceed very systematically. The most obvious possibility for a loop invariant J is to choose the postcondition $s = x * x$ because that will then clearly imply the postcondition since every formula is very good at implying itself. So let's use the following abbreviation and loop invariant:

$$\begin{aligned} \alpha &\overset{\text{def}}{\equiv} s := s + 2 * i + 1; i := i + 1 \\ J &\overset{\text{def}}{\equiv} s = x * x \end{aligned} \tag{2}$$

After this crucial choice, the rest of the proof steps are entirely systematic:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{s = 0, i = 0 \vdash J \qquad J, i < x \vdash [\alpha]J \quad J, \neg(i < x) \vdash s = x * x}{s = 0, i = 0 \vdash [\mathsf{while}(i < x)\,\alpha]s = x * x}\,\text{while}}{s = 0 \vdash [i := 0][\mathsf{while}(i < x)\,\alpha]s = x * x}\,[:=]_=}{s = 0 \vdash [i := 0; \mathsf{while}(i < x)\,\alpha]s = x * x}\,[;]}{\vdash [s := 0][i := 0; \mathsf{while}(i < x)\,\alpha]s = x * x}\,[:=]_=}{\vdash [s := 0; i := 0; \mathsf{while}(i < x)\,\alpha]s = x * x}\,[;]$$

Note that this proof cannot directly use axiom [:=] to substitute in the new value 0 for i because it still keeps changing in the loop. Instead rule $[:=]_=$ is used from Fig. 1, which keeps it around as an equational assumption $i = 0$ in the antecedent instead. For notational convenience, the above proof uses one optimization where the $[:=]_=$ step for $s := 0$ keeps using variable s instead of a new variable name y, because the context Γ, Δ is empty and the right hand side of the assignment does not mention s either.

While the proof of the right branch is entirely trivial by rule id with (2), the middle branch with the induction step poses quite a challenge, because (2) is not true after α even if it was true before, because the program α changes s while keeping x constant. So that loop invariant (2) was too naive. Instead, let's choose a loop invariant that says the same thing, just about the loop variable i instead of x because the result about the square of x is only attained in the end with partial progress till i:

$$J \overset{\text{def}}{\equiv} s = \boldsymbol{i} * \boldsymbol{i} \tag{3}$$

Thanks to our use of abbreviation J for the loop invariant that change does not change the structure of the above proof but gives us a new chance of proving its premises. Of course now the proof of the right premise becomes less trivial since id no longer suffices, but let's first worry about the middle branch that gave us so much trouble before.

The—most exciting—middle branch $J, i < x \vdash [\alpha]J$ can be proved using the usual decompositions with axioms (inside out for assignments) and the rule =R to replace the left hand side s of an equation $s = i * i$ with the right hand side $i * i$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{s = i*i, i < x \vdash i*i+2*i+1 = (i+1)*(i+1)}\,\mathbb{R}}{s = i*i, i < x \vdash s+2*i+1 = (i+1)*(i+1)}\,=\!R}{s = i*i, i < x \vdash [s := s+2*i+1](s = (i+1)*(i+1))}\,[:=]}{s = i*i, i < x \vdash [s := s+2*i+1][i := i+1](s = i*i)}\,[:=]}{s = i*i, i < x \vdash [s := s+2*i+1; i := i+1](s = i*i)}\,[;]}{J, i < x \vdash [\alpha]J}$$

But as soon as we march on to the right branch $J, \neg(i < x) \vdash s = x * x$, which is:

$$s = i * i, \neg(i < x) \vdash s = x * x$$

we find it impossible to prove, because it simply is not true. What could have gone wrong?

Of course. The loop invariant (3) was no good either. While it is inductive (the middle branch proves) it fails to imply the postcondition (the right branch does not). Contrast this with the loop invariant (2) which implies the postcondition but failed to be inductive. Neither are any good for proving the original DL formula. But if (3) is already inductive, then it might merely be missing additional knowledge.

When thinking back about where loop invariants came from (dropping $[\alpha^*]$) then the only information that could be missing in a loop invariant is to retain additional information about the past iterations that we still need to prove the postcondition. Indeed, (3) successfully relates the square variable s to the square of the loop variable i but doesn't tell us anything about how any of them relate to the input variable x. The loop guard tells us that $i < x$ holds when the loop body runs and that $\neg(i < x)$ holds when the loop exits. But it doesn't tell us that we indeed went about increasing i all the time until its value equals x. So let's discard the invariant candidate (3) and move on to:

$$J \overset{\text{def}}{\equiv} \boldsymbol{i \leq x} \wedge s = i * i \tag{4}$$

Having made that chance of loop invariant, the proof of the middle branch needs to be

adapted as indicated in boldface, but fortunately still proves:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\boldsymbol{i} \leq \boldsymbol{x}, s = i*i, i < x \vdash (\boldsymbol{i+1} \leq \boldsymbol{x} \wedge i*i+2*i+1 = (i+1)*(i+1))}\,\mathbb{R}}{\boldsymbol{i} \leq \boldsymbol{x}, s = i*i, i < x \vdash (\boldsymbol{i+1} \leq \boldsymbol{x} \wedge s+2*i+1 = (i+1)*(i+1))}\,=\!R}{\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i, i < x \vdash (\boldsymbol{i+1} \leq \boldsymbol{x} \wedge s+2*i+1 = (i+1)*(i+1))}\,\wedge L}{\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i, i < x \vdash [s := s+2*i+1](\boldsymbol{i+1} \leq \boldsymbol{x} \wedge s = (i+1)*(i+1))}\,[:=]}{\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i, i < x \vdash [s := s+2*i+1][i := i+1](\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i)}\,[:=]}{\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i, i < x \vdash [s := s+2*i+1; i := i+1](\boldsymbol{i} \leq \boldsymbol{x} \wedge s = i*i)}\,[;]}{J, i < x \vdash [\alpha]J}$$

The right branch $J, \neg(i < x) \vdash s = x*x$ can now be proved easily thanks to (4). The easiest way is a cut to show that $i = x$ follows by arithmetic from $i \leq x$ and $\neg(i < x)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{s = i*i, i = x \vdash s = i*i}\,\mathbb{R}}{s = i*i, i = x \vdash s = x*x}\,=\!R \qquad \dfrac{*}{i \leq x, \neg(i < x) \vdash i = x}\,\mathbb{R}}{i \leq x, s = i*i, \neg(i < x) \vdash s = x*x}\,\text{cut}}{i \leq x \wedge s = i*i, \neg(i < x) \vdash s = x*x}\,\wedge L}{J, \neg(i < x) \vdash s = x*x}$$

Finally the only thing that remains to be done is to prove the left branch $s = 0, i = 0 \vdash J$ (again there is a simple branch with trivial arithmetic on $0 = 0*0$):

$$\dfrac{\dfrac{\dfrac{s = 0, i = 0 \vdash 0 \leq x \qquad \dfrac{*}{\ldots \vdash 0 = 0*0}\,\mathbb{R}}{s = 0, i = 0 \vdash 0 \leq x \wedge 0 = 0*0}\,\wedge R}{s = 0, i = 0 \vdash i \leq x \wedge s = i*i}\,=\!R}{s = 0, i = 0 \vdash J}$$

Wait a second. We cannot prove that property. Shocking! What's wrong? Probably yet another mistake in the choice of the loop invariant J from (4). How could we change it?

It is quite crucial to realize that whatever we do to try and find another loop invariant, it won't succeed, simply because the formula $[\beta]\, s = x*x$ we were trying to prove is not valid. In fact the above proof attempt already points out what we're missing. We simply do not know if $0 \leq x$ is true in the beginning, but that is actually good news because the original DL formula is not even valid for negative x, so it cannot have a (sound) proof.

Obviously, all we need to do to get a proof now is to assume the precondition $x \geq 0$ in the beginning to obtain a provable DL formula and preserve this knowledge in the appropriate places of the proof:

$$x \geq 0 \rightarrow [s := 0; i := 0; \mathsf{while}(i < x)\, \{s := s+2*i+1; i := i+1\}]\, s = x*x$$

References

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, 2000.

[Pla15] André Platzer. Differential game logic. *ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015. doi:10.1145/2817824.

[Pla18] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Switzerland, 2018. URL: http://www.springer.com/978-3-319-63587-3.

[Pra76] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In *FOCS*, pages 109–121, Los Alamitos, 1976. IEEE. doi:10.1109/SFCS.1976.27.