1 Correctness of Coinductive Program Verification

The core of our approach is a general theorem about fixpoints, which is applied to program verification by giving a presentation of validity as a greatest fixpoint.

First we fix notation and recall standard definitions and results. Recall from Section 2.2 that our specifications are sets of claims from set $C \times \mathcal{P}(C)$, that is elements of $\mathcal{P}(C \times \mathcal{P}(C))$, where C is the set of configurations of the target language, and that an operational semantics $R \subseteq C \times C$ over configurations in C yields a function $\text{step}_R : \mathcal{P}(C \times \mathcal{P}(C)) \to \mathcal{P}(C \times \mathcal{P}(C))$. We can prove our main fixpoint theorem more generally, for any powerset lattice $(\mathcal{P}(D), \cup)$. This is the same setting as in the original Knaster-Tarski theorem [1], and like it our result can also be generalized to any complete lattice and even to a category, but our objective here is to keep it as simple and accessible as possible. Below, F and G range over monotone functions $\mathcal{P}(D) \to \mathcal{P}(D)$, and X, Y, A over sets in $\mathcal{P}(D)$. We lift union and subset pointwise on functions, so $(F \cup G)(X) = F(X) \cup G(X)$ and $F \subseteq G \equiv \forall X. F(X) \subseteq G(X)$. A fixpoint of F is a set A with $F(A) = A$. The Knaster-Tarski theorem implies that any monotone function F has a least and greatest fixpoint, resp., denoted μF and νF.

A defining property of least fixpoints is *induction*: $F(A) \subseteq A$ implies $\mu F \subseteq A$ for any set A. Dually, greatest fixpoints allow *coinduction*: $A \subseteq F(A)$ implies $A \subseteq \nu F$ for any set A. A set A is called *F-stable* if $A \subseteq F(A)$. A *closure operation* is a monotone function F satisfying the additional properties of being *extensive* $(\forall X, X \subseteq F(X))$ and *idempotent* $(\forall X, F(F(X)) = F(X))$.

First, we justify the name of the *F-closure* operation F^*:

Lemma 1. *If F is monotone then F^* is the least closure operator with $F \subseteq F^*$.*

Proof. First, F^* is a closure operator. For monotonicity, fix $A \subseteq B$. Then $F^*(A) \subseteq F^*(B)$ holds by induction. This is true since

$$F(F^*(B)) \cup A \subseteq F(F^*(B)) \cup B = F^*(B)$$

Then, we can use the definition of induction, instantiating $F_{ind} \equiv \lambda Y. F(Y) \cup A$ and $A_{ind} \equiv F^*(B)$, as well as noting that $\mu F_{ind} = F^*$.

Extensiveness holds by $X \subseteq F(F^*(X)) \cup X = F^*(X)$. For idempotence it suffices, given extensiveness, to show $F^*(F^*(X)) \subseteq F^*(X)$, which follows by induction from

$$F(F^*(X)) \cup F^*(X) = F(F^*(X)) \cup F(F^*(X)) \cup X = F^*(X)$$

with $F_{ind} \equiv \lambda Y. F(Y) \cup F^*(X)$ and $A_{ind} = F^*(X)$. Next, by monotonicity of F and extensiveness of F^*, F^* is above F by $F(X) \subseteq F(F^*(X)) \subseteq F(F^*(X)) \cup X = F^*(X)$.

It is smallest since if $F \subseteq G$ for closure operator G, $F^* \subseteq G$ by induction from $F(G(X)) \cup X \subseteq G(G(X)) \cup G(X) \subseteq G(X)$, so F^* is the least closure operator above F. □

Corollary 1. *The operation $-^*$ is monotone. For monotone F and G, if $F \subseteq G$, then G^* is a closure operator with $F \subseteq G \subseteq G^*$, so $F^* \subseteq G^*$.*

Now we are ready to state and prove our key coinduction theorem.

Theorem 1. *If F and G are monotone, and $G(F(A)) \subseteq F(G^*(A))$ for any A, then $X \subseteq F(G^*(X))$ implies $X \subseteq \nu F$ for any X.*

Proof. First, note from the assumptions we have that $G(F(G^*(X))) \subseteq F(G^*(G^*(X)))$, that is, $G(F(G^*(X))) \subseteq F(G^*(X))$ by idempotence. We also have $X \subseteq F(G^*(X))$ by assumption, so we have $G(F(G^*(X))) \cup X \subseteq F(G^*(X))$. Then, $G^*(X) \subseteq F(G^*(X))$ follows by the definition of induction, instantiating $F_{ind} \equiv \lambda Y.G(Y) \cup X$ and $A_{ind} \equiv F(G^*(X))$, noting that $\mu F_{ind} = G^*$. Finally, coinduction implies $G^*(X) \subseteq \nu F$, and extensiveness implies $X \subseteq \nu F$. □

In our program verification setting with languages given as relations $R \subseteq C \times C$, F is usually step_R. To verify programs, we express valid_R as a greatest fixpoint:

Lemma 2. $\text{valid}_R = \nu\, \text{step}_R$

Proof. First, let's recall the definition of step_R:

$$\text{step}_R(S) = \{(c, P) \mid c \in P \ \vee \ \exists d\,.\, c \to_R d \wedge (d, P) \in S\}$$

To prove the lemma, we first show valid_R is step_R-stable, or that $\text{valid}_R \subseteq \text{step}_R(\text{valid}_R)$. Suppose $(c, P) \in \text{valid}_R$. If $c \in P$ then $(c, P) \in \text{step}_R(\text{valid}_R)$, else c has a successor d which either starts an infinite path in R or reaches an element of P. In either case, $(d, P) \in \text{valid}_R$ so $(c, P) \in \text{step}_R(\text{valid}_R)$.

Now, we show valid_R is the largest step_R-stable set. Fix $S \subseteq \text{step}_R(S)$ and let $(c, P) \in S$. If there is an infinite path in R beginning at c then $(c, P) \in \text{valid}_R$. Otherwise, R is well-founded on the set of configurations reachable from c. Then by well-founded induction $(d, P) \in \text{valid}_R$ for any d with $c \to_R d$ and $(d, P) \in S$. If $c \in P$ then $(c, P) \in \text{valid}_R$, else by $(c, P) \in S \subseteq \text{step}_R(S)$ and the induction hypothesis there is a successor d of c with $(d, P) \in \text{valid}_R$, and thus also $(c, P) \in \text{valid}_R$. □

1.1 Derived Rules

Instantiating Theorem 1 involves choices for F and G. In the examples from Section 3, we chose both F and G to be step_R. However, step_R may not always be the best choice for G. In general, as long as the conditions for Theorem 1 are satisfied, G can be anything. Specifically, fixing F, we can see that the set of functions G that meet the condition $G(F(X)) \subseteq F(G^*(X))$ for all X includes F and is closed under union. This motivates the following:

Definition 1. *Given monotone functions F and G, we say that G is a* valid derived rule for F *whenever $G(F(X)) \subseteq F((F \cup G)^*(X))$ for all X.*

Motivated by this definition, we have the following corollary of Theorem 1:

Corollary 2. *If G is a valid derived rule for F, then $X \subseteq F((F \cup G)^*(X))$ implies $X \subseteq \nu F$.*

Proof. First, $F(F(X)) \subseteq F((F \cup G)^*(X))$. This along with the assumption implies $(F \cup G)(F(X)) \subseteq F((F \cup G)^*(X))$. Then we can apply Theorem 1 with G instantiated to $F \cup G$ to prove the claim. □

Now we give two examples: a rule for transitivity inspired by our definition for sequential composition, and a rule for modularity:

As we saw in Section 2.2, when we turn this theorem to program verification G will play the role of acceptable proof rules, so we would like to define it compositionally. Note that $\forall X.\, F(F(X)) \subseteq F(F^*(X))$ and that If both $\forall X.\, G(F(X)) \subseteq F(G^*(X))$ and $\forall X.\, H(F(X)) \subseteq F(G^*(X))$ then also $(G \cup H)(F(X)) \subseteq F((G \cup H)^*(X))$. So, we say that G is a *sound rule for* F if $\forall X.\, G(X) \subseteq F((G \cup F)^*(X))$.

Lemma 3. trans *defined by* $\mathrm{trans}(S) = \{(c, P) \mid \exists Q.(c, Q) \in S \wedge \forall d \in Q, (d, P) \in S\}$ *is a valid derived rule for* step_R.

Proof. Suppose $(c, P) \in \mathrm{trans}(\mathrm{step}_R(S))$. Fix Q from the definition $(c, Q) \in \mathrm{step}_R(S)$, so either $c \in Q$ or $(c', Q) \in S$ for some c' with $c \rightarrow_R c'$. In the first case let d be c to conclude $(c, P) \in \mathrm{step}_R(S) \subseteq \mathrm{step}_R((\mathrm{trans} \cup \mathrm{step}_R)^*(S))$. Otherwise we use $\forall d \in Q, (d, P) \in \mathrm{step}_R(S)$ to conclude $(c', P) \in \mathrm{trans}(S \cup \mathrm{step}_R(S))$, and thus $(c, P) \in \mathrm{step}_R(\mathrm{trans}(S \cup \mathrm{step}_R(S))) \subseteq \mathrm{step}_R((\mathrm{trans} \cup \mathrm{step}_R)^*(S))$. □

This derived rule turns out to be very useful for our Coq proofs in Section 4.

Lemma 4. *Function* proved_R *on sets of claims defined by* $\mathrm{proved}_R(S) = S \cup \mathrm{valid}_R$ *is a valid derived rule for* step_R.

Proof. If $(c, P) \in \mathrm{proved}_R(\mathrm{step}_R(S))$ either $(c, P) \in \mathrm{valid}_R = \mathrm{step}_R(\mathrm{valid}_R)$ or $(c, P) \in \mathrm{step}_R(S)$. Note $\mathrm{valid}_R \cup S = \mathrm{proved}_R(S) \subseteq (\mathrm{proved}_R \cup \mathrm{step}_R)^*(S)$. □

Another approach to modular proof uses monotonicity properties. An inclusion of the form $A \subseteq \mathrm{step}_R((G \cup \mathrm{step}_R)^*(B))$ with G a valid derived rule for step_R can be seen as a proof of specification A with unproved assumptions $B \setminus A$. For example, proving a functions's public specification and loop lemmas assuming some claims about library functions. Two such facts compose by monotonicity properties to give $(A_1 \cup A_2) \subseteq \mathrm{step}_R(((G_1 \cup G_2) \cup \mathrm{step}_R)^*(B_1 \cup B_2))$, proving a larger set $A_1 \cup A_2$ under a potentially smaller set of remaining assumptions $(B_1 \cup B_2) \setminus (A_1 \cup A_2)$. If enough partial arguments are combined that $B \subseteq A$ then Theorem 1 shows the validity of the combined claims.

1.2 Proof of Sequential Composition Lemma

Below is a proof of the lemma connecting claims via sequential composition, seen in Section 3.2.

Lemma 5. $S_1 \fatsemi S_2 \subseteq \mathrm{valid}_R$ *if* $S_1 \subseteq \mathrm{valid}_R$ *and* $S_2 \subseteq \mathrm{valid}_R$.

Proof. Let $(c, P) \in S_1 \fatsemi S_2$. We want to show $c \Rightarrow_R P$, i.e. that c reaches a state in P or loops infinitely. Since $(c, P) \in S_1 \fatsemi S_2$, there is a Q such that $(c, Q) \in S_1$. Since $S_1 \subseteq \text{valid}_R$, either c loops forever (and we are done) or c reaches some state, say x, in Q. But then we also know that for all d in Q, $(d, P) \in S_2$. Finally, $S_2 \subseteq \text{valid}_R$ implies x loops infinitely (which implies c loops infinitely and we are done), or x reaches some state in P, which implies c reaches some state in P as needed. □

1.3 Relative Completeness

Normally, a relative completeness argument requires a lot of mathematical machinery. Generally, the notion of validity needs to be encoded using Gödel numbering or something similar. One can see examples of this in Cook's [2] or Winskel's [3] proof of relative completeness of Hoare logic, or even in the proof for relative completeness of the more general reachability logic framework [4].

We are not presenting a syntactic proof system, so relative completeness for us requires a different formulation. To prove the validity of a desired set X of claims in our system, it is in general necessary to apply Theorem 1 on a larger set S of claims, just as it is in general necessary to introduce loop invariants or other intermediate assertions within a proof in Hoare logic.

For relative completeness, we want to show that any valid claim (or set of valid claims) can be proved valid using our coinductive verification framework. We do this by showing that for any set of valid claims, we can prove a larger set S of claims is derivable in our framework.

So, for any set X of valid claims and any G, we can take as S the set valid_R. The proof of Lemma 2 established that $\text{valid}_R \subseteq \text{step}_R(\text{valid}_R)$. Thus, from extensiveness and monotonicity of step_R we can see that

$$\begin{aligned}
&\text{valid}_R \subseteq G^*(\text{valid}_R) && \text{(extensiveness)}\\
&\Rightarrow \text{step}_R(\text{valid}_R) \subseteq \text{step}_R(G^*(\text{valid}_R)) && \text{(monotonicity)}\\
&\Rightarrow \text{valid}_R \subseteq \text{step}_R(G^*(\text{valid}_R))
\end{aligned}$$

This is exactly the hypothesis of Theorem 1. This shows that the entire set valid_R is derivable from our proof system. Thus, showing our original set of claims X is valid reduces to simply showing that $X \subseteq S = \text{valid}_R$, which is immediate by the definition of valid_R.

One might object that this is trivial, but then they should object all the more about a conventional relative completeness result. A general purpose specification language is undecidable so any complete proof system necessarily has a rule with a hypothesis referring to a semantic truth in the predicate language. A relative completeness argument consists of showing how to tediously Gödel-encode the desired notion of validity into the predicate language, and then showing that the rules of the proof system are strong enough to make use of such a complicated predicate. We obtain an equally strong result.

2 Reachability Logic is Coinduction

In this section, we give a proof of the main lemma needed for proving reachability logic is an instance of coinduction. Recall the statement:

Lemma 6 (5.3)**.** *Given a proof derivation of $\mathcal{A} \vdash_{\mathcal{C}} \varphi_a \Rightarrow \varphi_b$ which does not use the Circularity proof rule (last rule in Figure 6), if $R \vDash^{+} \mathcal{A}$ and $\mathcal{C}$ is nonempty then $S_{\varphi_a \Rightarrow \varphi_b} \subseteq \text{step}_R(\text{derived}^*_R(\overline{\mathcal{C}}))$.*

We now give the definition $\text{derived}_R(X) = \text{step}_R(X) \cup \text{trans}(X) \cup \text{proved}_R(X)$, which is a valid derived rule as defined in Section 1.1.

The proof will proceed by structural induction on the Reachability Logic derivation, but the statement must be strengthened for the induction argument to succeed. We will need to use the idea of a reachability rule holding "with progress", which also contributed to the definition of $R \vDash^{+} \mathcal{A}$. To express this notion in our coinductive setting we a definition

$$\text{next}_R(X) = \{(x, P) \mid \exists x'.x \rightarrow_R x' \wedge (x', P) \in X\}$$

In terms of this new definition, $R \vDash^{+} \mathcal{A}$ iff $\overline{\mathcal{A}} \subseteq \text{next}_R(\text{valid}_R)$. Note that $\text{next}_R(X) \subseteq \text{step}_R(X)$ for any transition relation R and set of claims X. We must also address derivations with empty $\mathcal{C}$.

Claim 1. *Given sets of reachability rules $\mathcal{A}_0$ and $\mathcal{C}_0$, a transition relation R such that $R \vDash^{+} \mathcal{A}_0$, and a reachability logic derivation of $\mathcal{A} \vdash_{\mathcal{C}} \varphi_a \Rightarrow \varphi_b$ which does not use Circularity, then*

$$S_{\varphi_a \Rightarrow \varphi_b} \subseteq \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0})) \text{ holds if } \mathcal{A} = \mathcal{A}_0 \text{ and } \mathcal{C} = \mathcal{C}_0$$
$$S_{\varphi_a \Rightarrow \varphi_b} \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0}) \text{ holds if } \mathcal{A} = \mathcal{A}_0 \cup \mathcal{C}_0 \text{ and } \mathcal{C} \text{ is empty.}$$

This strengthened claim is proved by induction over the derivation of $\mathcal{A} \vdash_{\mathcal{C}} \varphi_a \Rightarrow \varphi_b$.

2.1 Axiom

In the axiom rule we know that $\varphi_a \Rightarrow \varphi_b$ is in $\mathcal{A}_0$ or $\mathcal{C}_0$. If the rule is in $\mathcal{A}_0$ then we have

$$S_{\varphi_a \Rightarrow \varphi_b} \subseteq \overline{\mathcal{A}_0} \subseteq \text{next}_R(\text{valid}_R) \subseteq \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0}))$$

If the rule is in $\mathcal{C}_0$ but not in $\mathcal{A}_0$ then we know $\mathcal{C}$ is empty, so it suffices to show

$$S_{\varphi_a \Rightarrow \varphi_b} \subseteq \overline{\mathcal{C}_0} \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0})$$

2.2 Reflexivity

This rule may only been used when $\mathcal{C}$ is empty, so it suffices to show $S_{\varphi \Rightarrow \varphi} \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0})$.
If $(c, P) \in S_{\varphi \Rightarrow \varphi}$, then by definition $c \in P$. Therefore $(c, P) \in \text{step}_R(\emptyset) \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0})$.

2.3 Transitivity

First observe that $S_{\varphi_1 \Rightarrow \varphi_3} \subseteq (S_{\varphi_1 \Rightarrow \varphi_2} \fatsemi S_{\varphi_2 \Rightarrow \varphi_3})$. Also note that the second hypothesis of the transitivity rule always has $\mathcal{A} = \mathcal{A}_0 \cup \mathcal{C}_0$ and empty $\mathcal{C}$, so $S_{\varphi_2 \Rightarrow \varphi_3} \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0})$. Now consider cases on $\mathcal{C}$.

If $\mathcal{C}$ is empty, then we have $S_{\varphi_1 \Rightarrow \varphi_2} \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0})$ and calculate

$$\begin{aligned} S_{\varphi_1 \Rightarrow \varphi_3} &\subseteq S_{\varphi_1 \Rightarrow \varphi_2} \fatsemi S_{\varphi_2 \Rightarrow \varphi_3} \\ &\subseteq \text{derived}^*_R(\overline{\mathcal{C}_0}) \fatsemi \text{derived}^*_R(\overline{\mathcal{C}_0}) \\ &= \text{trans}(\text{derived}^*_R(\overline{\mathcal{C}_0})) \subseteq \text{derived}^*_R(\overline{\mathcal{C}_0}) \end{aligned}$$

If $\mathcal{C}$ is $\mathcal{C}_0$, then we have the stronger assumption $S_{\varphi_1 \Rightarrow \varphi_2} \subseteq \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0}))$, but also a stronger obligation. Using the fact that $(\text{next}_R(X) \fatsemi Y) = \text{next}_R(X \fatsemi Y)$, we calculate

$$\begin{aligned} S_{\varphi_1 \Rightarrow \varphi_3} &\subseteq S_{\varphi_1 \Rightarrow \varphi_2} \fatsemi S_{\varphi_2 \Rightarrow \varphi_3} \\ &\subseteq \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0})) \fatsemi \text{derived}^*_R(\overline{\mathcal{C}_0}) \\ &= \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0}) \fatsemi \text{derived}^*_R(\overline{\mathcal{C}_0})) \\ &= \text{next}_R(\text{trans}(\text{derived}^*_R(\overline{\mathcal{C}_0}))) \subseteq \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0})) \end{aligned}$$

2.4 Consequence

In matching logic $\varphi \to \varphi'$ holds if and only if $\overline{\rho}(\varphi) \subseteq \overline{\rho}(\varphi')$. For any claim $(c, P) \in S_{\varphi_1 \Rightarrow \varphi_2}$ there is some ρ such that $P = \overline{\rho}(\varphi_2)$ and $c \in \overline{\rho}(\varphi_1)$. By the assumptions of this proof rule we conclude that $c \in \overline{\rho}(\varphi'_1)$ and that $\overline{\rho}(\varphi'_2)$ is a subset of P. Thus there is a claim $(c, P') \in S_{\varphi'_1 \Rightarrow \varphi'_2}$ for some $P' \subseteq P$.

If $\mathcal{C}$ is empty, then $(c, P') \in \text{derived}^*_R(\overline{\mathcal{C}_0})$. By using trans and the fact that any claim (x, P) with $x \in P'$ is trivially valid, $(c, P) \in \text{derived}^*_R(\overline{\mathcal{C}_0})$.

If $\mathcal{C}$ is nonempty, then $(c, P') \in \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0}))$. By the definition of next_R, there is a configuration d with $c \to_R d$ and $(d, P') \in \text{derived}^*_R(\overline{\mathcal{C}_0})$. By the argument from the other case $(d, P) \in \text{derived}^*_R(\overline{\mathcal{C}_0})$, so $(c, P) \in \text{next}_R(\text{derived}^*_R(\overline{\mathcal{C}_0}))$.

2.5 Remaining Rules

For the remaining reachability logic proof rules, the hypotheses have the same sets $\mathcal{A}$ and $\mathcal{C}$ as the conclusion of the rule. Therefore the induction hypotheses and the statement to be shown are all inclusions with the same set of claims on the right hand side. In this situation it suffices to show that any claim (c, P) in the set $S_{\varphi_a \Rightarrow \varphi_b}$ is also contained in one of the sets $S_{\varphi_c \Rightarrow \varphi_d}$ appearing in the induction hypotheses. This avoids considering cases on $\mathcal{C}$.

2.6 Logical Framing

This rule requires ψ be a patternless matching logic formula, which means it always evaluates to either the total set or the empty set. By this and the

semantics of conjunction in matching logic,

$$\begin{aligned} S_{\varphi \wedge \psi \Rightarrow \varphi' \wedge \psi} &= \{(c, \overline{\rho}(\varphi') \cap \overline{\rho}(\psi)) \mid c \in \overline{\rho}(\varphi) \cap \overline{\rho}(\psi),\ \forall \rho : \mathit{Var} \to \mathit{Cfg}\} \\ &= \{(c, \overline{\rho}(\varphi') \mid c \in \overline{\rho}(\varphi),\ \forall \rho : \mathit{Var} \to \mathit{Cfg} \text{ s.t. } \overline{\rho}(\psi) = \mathit{Cfg}\} \subseteq S_{\varphi \Rightarrow \varphi'} \end{aligned}$$

2.7 Abstraction

In this rule, the free variables of φ' are disjoint from X. By definition, $c \in \overline{\rho}(\exists X \varphi)$ holds iff there is some ρ' such that $c \in \overline{\rho'}(\varphi)$ and ρ' agrees with ρ on variables other than those in X. This required agreement implies $\overline{\rho'}(\varphi') = \overline{\rho}(\varphi)$. Therefore $S_{\exists X \varphi \Rightarrow \varphi'} = S_{\varphi \Rightarrow \varphi'}$.

2.8 Case Analysis

By the semantics of disjunction in matching logic, $\overline{\rho}(\varphi_1 \vee \varphi_2) = \overline{\rho}(\varphi_1) \cup \overline{\rho}(\varphi_2)$ for any ρ. If $(c, \overline{\rho}(\varphi)) \in S_{\varphi_1 \vee \varphi_2 \Rightarrow \varphi}$, then $c \in \overline{\rho}(\varphi_1)$ or $c \in \overline{\rho}(\varphi_2)$ and therefore $(c, \overline{\rho}(\varphi)) \in S_{\varphi_1 \Rightarrow \varphi} \cup S_{\varphi_2 \Rightarrow \varphi}$. □

References

[1] Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics **5**(2) (1955) 285–309

[2] Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. **7**(1) (1978) 70–90

[3] Winskel, G.: The Formal Semantics of Programming Languages: An Introduction. MIT Press, Cambridge, MA, USA (1993)

[4] Roşu, G., Ştefănescu, A., Ciobâcă, Ş., Moore, B.M.: One-path reachability logic. In: LICS, IEEE (2013) 358–367