The semantics of the Maude strategy language

Technical report SIC 21/01 — August, 2021

Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, Alberto Verdejo
Facultad de Informática, Universidad Complutense de Madrid, Spain
{rubenrub,narciso,ipandreu,jalberto}@ucm.es

1 Strategy language syntax

Given a rewrite theory $\mathcal{R} = (\Sigma, E \uplus Ax, R)$ and a set Ω of strategy identifiers, we define $\text{Strat}_{\mathcal{R},\Omega}$ as the set of terms generated from the non-terminal *Strat* of the following grammar:

⟨*Strat*⟩ ::= `idle` | `fail`
| ⟨*BasicStrat*⟩
| `top(` ⟨*BasicStrat*⟩ `)`
| ⟨*Strat*⟩ `?` ⟨*Strat*⟩ `:` ⟨*Strat*⟩
| ⟨*Test*⟩
| ⟨*Strat*⟩ `;` ⟨*Strat*⟩
| ⟨*Strat*⟩ `|` ⟨*Strat*⟩
| ⟨*Strat*⟩ `*`
| ⟨*Matchrew*⟩
| ⟨*SCall*⟩
| `one(` ⟨*Strat*⟩ `)`

⟨*BasicStrat*⟩ ::= ⟨*Label*⟩ | `all`
| ⟨*Label*⟩ [`[` ⟨*Substitution*⟩ `]`] [`{` ⟨*StratList*⟩ `}`]

⟨*StratList*⟩ ::= ⟨*Strat*⟩
| ⟨*Strat*⟩ `,` ⟨*StratList*⟩

⟨*Substitution*⟩ ::= ⟨*Variable*⟩ `<-` ⟨*Term*⟩
| ⟨*Substitution*⟩ `,` ⟨*Substitution*⟩

⟨*SCall*⟩ ::= ⟨*SLabel*⟩
| ⟨*SLabel*⟩ `(` ⟨*TermList*⟩ `)`

⟨*TermList*⟩ ::= ⟨*Term*⟩
| ⟨*Term*⟩ `,` ⟨*TermList*⟩

where *Term* can be any element of $T_\Sigma(X)$, *Label* any label in R, *SLabel* any element of Ω and *Type* any type of the rewriting theory.

⟨*TestVariant*⟩ ::= `amatch`
| `match`
| `xmatch`

⟨*MrewVariant*⟩ ::= `amatchrew`
| `matchrew`
| `xmatchrew`

⟨*Condition*⟩ ::= ⟨*Term*⟩ `=` ⟨*Term*⟩
| ⟨*Term*⟩ `:=` ⟨*Term*⟩
| ⟨*Term*⟩ `:` ⟨*Type*⟩

⟨*RuleCondition*⟩ ::= ⟨*Condition*⟩
| ⟨*Term*⟩ `=>` ⟨*Term*⟩

⟨*Test*⟩ ::= ⟨*TestVariant*⟩ ⟨*Term*⟩ [`s.t.` ⟨*Condition*⟩]

⟨*Matchrew*⟩ ::= ⟨*MrewVariant*⟩ ⟨*Term*⟩ [`s.t.` ⟨*Condition*⟩] `by` ⟨*VarStratList*⟩

⟨*VarStratList*⟩ ::= ⟨*Variable*⟩ `using` ⟨*Strat*⟩
| ⟨*Variable*⟩ `using` ⟨*Strat*⟩ `,` ⟨*VarStratList*⟩

While the previous constructions contain all the expressiveness of the language, some meaningful derivatives can be defined.

⟨*Strat*⟩ ::= […]
| ⟨*Strat*⟩ `+` — $\alpha^+ \equiv \alpha\ ;\ \alpha^*$
| ⟨*Strat*⟩ `or-else` ⟨*Strat*⟩ — α `or-else` $\beta \equiv \alpha$ `?` `idle` `:` β
| `not(` ⟨*Strat*⟩ `)` — `not(`α`)` $\equiv \alpha$ `?` `fail` `:` `idle`
| ⟨*Strat*⟩ `!` — α`!` $\equiv \alpha^*$ `;` `not(`α`)`
| `try(` ⟨*Strat*⟩ `)` — `try(`α`)` $\equiv \alpha$ `?` `idle` `:` `idle`
| `test(` ⟨*Strat*⟩ `)` — `test(`α`)` $\equiv$ `not(not(`α`))`

A strategy module starts with `smod` and ends with `endsm`, and it can import other modules and contain strategy declarations and definitions:

⟨*StratDecl*⟩ ::= `strat` ⟨*SLabel*⟩+ `@` ⟨*Type*⟩ .
| `strat` ⟨*SLabel*⟩+ `:` ⟨*Type*⟩* `@` ⟨*Type*⟩ .

⟨*StratDef*⟩ ::= `sd` ⟨*SCall*⟩ `:=` ⟨*Strat*⟩ .
| `csd` ⟨*SCall*⟩ `:=` ⟨*Strat*⟩ `if` ⟨*Condition*⟩ .

Strategy modules are encoded in two sets: the set of *named strategies* or *strategy declarations* Ω and the set of *strategy definitions* D. Like the symbols of the signature, strategies can be overloaded by the type of their arguments so that each named strategy is identified by both its label and its argument kinds. The strategy definitions can be seen as $D \subseteq \mathit{label}(\Omega) \times T_\Sigma(X)^* \times \mathrm{Strat}_{\mathcal{R},\Omega} \times \mathit{Condition}$. Some restrictions on types, arities and scopes must be checked:

- For each strategy call $sl(t_1, \dots, t_n)$ there must be a strategy declaration in Ω with label sl and argument kinds $k_1, \dots, k_m$ such that $m = n$ and $t_i \in T_{\Sigma,k_i}(X)$ for all i between 1 and n.
- Two strategy declarations whose arguments fall in the same connected components declare the same named strategy, as in Σ. It usually does not make sense to write two declarations in that situation. However, it may be useful, for example, to extend the argument or subject type of the strategy in an extending module.
- All variable occurrences must be bound except when they appear in the pattern of a `match` or `matchrew` construct, in the left-hand side of an assignment condition fragment, or in the left-hand side of a strategy definition.

  Variables in a pattern are bound in the corresponding condition and in each `matchrew`'s substrategy. Variables in assignment condition fragments are bound in the following fragments and in the `matchrew`'s substrategies, if applicable. Variables in the *VarStratList* of `matchrew` must be distinct and appear in its pattern. Any other variable in strategy definitions must appear in the left-hand side or in the condition.

2 Semantics infrastructure

The execution of a strategy, like free rewriting, is not necessarily deterministic. The number of solutions yielded by the strategy may be zero if the strategy fails, any positive number, or even an infinite one. Moreover, as recursion is allowed in strategy modules, the rewriting process may not terminate, even if it has already produced some results. Hence, if we are interested in describing the results of a strategy computation, we are bound to use sets of terms with an explicit representation of non-termination. This motivates the following definition:

▸ **Definition.** For any set M we define

$$\mathcal{P}_\perp(M) := \frac{\mathcal{P}(M \cup \{\perp\})}{\sim} \qquad A \sim B \iff A = B \vee (A \text{ is not finite and } A \oplus B = \{\perp\})$$

The additional element $\perp$ indicates non-termination or temporary undefinedness, and $\oplus$ is the symmetric difference of two sets. The equivalence relation $\sim$ states that we do not care about infinite sets containing $\perp$, because the calculation of an infinite set of solutions never terminates. In simpler words, $\mathcal{P}_\perp(M)$ is $\mathcal{P}(M)$ with an additional copy of the finite sets, marked by $\perp$. In practice, we will usually confuse equivalence classes and sets, writing $\perp \in A$ if A contains $\perp$ or A is infinite. Unions are well defined and $A \backslash \{\perp\}$ is always an unambiguous subset of M.

In our semantic setting, sets containing $\perp$ (or infinite sets) are mean to be still open in the sense that the undefinedness behind $\perp$ can be solved during execution to add further results. On the contrary, sets without $\perp$ are supposed to be already complete. This idea of *extension* of solution sets gives sense to the following order relation:

▸ **Definition.** For any set M, the order relation $\leq$ in $\mathcal{P}_\perp(M)$ is defined as

$$A \leq B \iff \begin{cases} A \backslash \{\perp\} \subseteq B & \text{if } \perp \in A \\ A = B & \text{if } \perp \notin A \end{cases}$$

In Appendix A.1, we prove $(\mathcal{P}_\perp(M), \leq)$ is a *chain-complete partially ordered set*, whose minimum is $\{\perp\}$. Directly dealing with this indeterminate symbol is cumbersome, so we define the following notation:

▸ **Definition.** For any sets M and N, $A \in \mathcal{P}_\perp(M)$ and $B_x \in \mathcal{P}_\perp(N)$ for every $x \in M$, we define

$$\mathsf{let}\ x \leftarrow\!\!\prec A : B_x := \begin{cases} \{\perp\} \cup \bigcup_{x \in A \setminus \{\perp\}} B_x & \text{if } \perp \in A \\ \bigcup_{x \in A} B_x & \text{if } \perp \notin A \end{cases}$$

Formally, let is a function $\mathcal{P}_\perp(M) \times (M \to \mathcal{P}_\perp(N)) \to \mathcal{P}_\perp(N)$ where $B(x) = B_x$.

2.1 Basic operations of rewriting logic

Given a rewrite theory $\mathcal{R} = (\Sigma, E \uplus Ax, R)$ with set of variables X and strategies given by (Ω, D), the semantic function for a strategy $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$ can take the form

$$[\![\alpha]\!] : (X \to T_\Sigma(X)) \times T_\Sigma(X) \to \mathcal{P}_\perp(T_\Sigma(X))$$

It receives a substitution and a term, and produce the set of all terms obtained by rewriting according to α, including $\perp$ if the calculation does not terminate. To improve legibility we write VEnv for $X \to T_\Sigma(X)$ and call this functional class SFun and its domain SDom. Using the let function introduced above, the composition of semantic functions can be defined

$$\begin{array}{rccc} \circ : & \mathrm{SFun} \times \mathrm{SFun} & \to & \mathrm{SFun} \\ & (g, f) & \mapsto & g \circ f \end{array} \qquad (g \circ f)(\theta, t) = \mathsf{let}\ t' \leftarrow\!\!\prec f(\theta, t) : g(\theta, t')$$

And we write for any $f \in \mathrm{SFun}$, $f^0(\theta, t) = \{t\}$ and $f^{n+1} = f \circ f^n$. The composition of substitutions $\sigma_2 \circ \sigma_1$ is defined by $(\sigma_2 \circ \sigma_1)(x) = \overline{\sigma}_2(\sigma_1(x))$ where $\overline{\sigma} : T_\Sigma(X) \to T_\Sigma(X)$ is the inductive extension to terms of a substitution σ. The property $\overline{\sigma_2 \circ \sigma_1} = \overline{\sigma}_2 \circ \overline{\sigma}_1$ holds in the usual functional sense.

In order to properly define the various semantics in this document, we have to refer to some auxiliary functions for matching and checking conditions. We assume matching functions are already defined

$$\mathsf{match} : T_\Sigma(X) \times T_\Sigma(X) \to \mathcal{P}(\mathrm{VEnv})$$

The basic $\mathsf{match}(p, t)$ provides a (finite) set of substitutions such that the pattern p matches t on top. Variables which do not occur in p are mapped to themselves (i.e. are not bound). For matching anywhere or with extension, the function result also includes a *context*

$$\mathsf{amatch}, \mathsf{xmatch} : T_\Sigma(X) \times T_\Sigma(X) \to \mathcal{P}(\mathrm{VEnv} \times T_\Sigma(X))$$

The context $c \in T_\Sigma(X)$ is a term with single special fresh variable $\ominus$ occurring once, called the *hole*. We will usually use c as a function that replaces the hole by its argument.

Conditions syntax is defined in [1, §19.1]. Since rewriting conditions can now be controlled by strategies, we need to redefine condition checking. The function

$$\mathsf{check} : \mathit{RuleCondition} \times \mathrm{VEnv} \times \mathrm{Strat}^*_{\mathcal{R},\Omega} \times \mathrm{VEnv} \to \mathcal{P}_\perp(\mathrm{VEnv})$$

will provide the set of (most general) substitutions $\mathsf{check}(C, \theta, \alpha_1 \cdots \alpha_m, \theta_s)$ such that the condition C holds preserving the bindings from θ and using α_1 to α_m (whose variables are bound by θ_s) to restrict rewriting in the m rewriting condition fragments of C. That reads the recursive definition below

$$\begin{aligned} \mathsf{check}(\mathtt{true}, \theta; \overline{\alpha}, \theta_s) &= \{\theta\} \\ \mathsf{check}(l = r \wedge C, \theta; \overline{\alpha}, \theta_s) &= \mathsf{check}(C, \theta; \overline{\alpha}, \theta_s) \textbf{ if } \theta(l) = \theta(r) \textbf{ else } \emptyset \\ \mathsf{check}(t : s \wedge C, \theta, \overline{\alpha}, \theta_s) &= \mathsf{check}(C, \theta; \overline{\alpha}, \theta_s) \textbf{ if } \theta(t) \in T_{\Sigma,s}(X) \textbf{ else } \emptyset \\ \mathsf{check}(l := r \wedge C, \theta; \overline{\alpha}, \theta_s) &= \cup \{\mathsf{check}(C, \sigma \circ \theta, \overline{\alpha}, \theta_s) : \sigma \in \mathsf{xmatch}(\theta(l), \theta(r))\} \\ \mathsf{check}(l \Rightarrow r \wedge C, \theta; \alpha\overline{\alpha}, \theta_s) &= \mathsf{let}\ t \leftarrow\!\!\prec [\![\alpha]\!](\theta_s, \theta(l)) : \cup \{\mathsf{check}(C, \sigma \circ \theta; \overline{\alpha}, \theta_s) : \sigma \in \mathsf{xmatch}(\theta(r), t)\} \end{aligned}$$

For this definition to make sense, m must be greater or equal than the number of rewriting condition fragments in C. The function $\mathrm{nrewf} : \mathit{RuleCondition} \to \mathbb{N}$ denotes this number. When applying this function to an equational condition, when $\mathrm{nrewf}(C) = 0$, $\perp$ will not appear and strategies are not needed. Then we drop the last two parameters.

$$\mathsf{check} : \mathit{Condition} \times \mathrm{VEnv} \to \mathcal{P}_\perp(\mathrm{VEnv}) \qquad \mathsf{check}(C, \theta) = \mathsf{check}(C, \theta; \varepsilon, \mathrm{id})$$

Matching and condition checks often appear together, so the following definitions will be used for readability and space-saving. Their signature is

$$T_\Sigma(X) \times T_\Sigma(X) \times \mathit{RuleCondition} \times \mathrm{VEnv} \times \mathrm{Strat}^*_{\mathcal{R},\Omega} \times \mathrm{VEnv} \to \mathcal{P}_\perp(\mathrm{VEnv})$$

where

$$\mathsf{mcheck}(p,t,C,\theta;\overline{\alpha},\theta_s) = \bigcup_{\sigma \in \mathsf{match}(\theta(p),t)} \mathsf{check}(C,\sigma\circ\theta;\overline{\alpha},\theta_s)$$

amcheck is defined similarly but adding the context to the result. As usual, we drop the strategies when not needed. With all this we can define rule application as

$$\mathsf{ruleApply}(t,rl,\theta;\alpha_1\cdots\alpha_n,\theta_s) = \bigcup_{\substack{(rl,l,r,C)\in R\\ \mathrm{nrewf}(C)=n}} \mathsf{let}\ (\sigma,c) \leftarrowtail \mathsf{amcheck}(l,t,C,\theta;\alpha_1\cdots\alpha_n,\theta_s) : \{c(\sigma(r))\},$$

changing amcheck by mcheck if only applying on top.

Our last shortcut definition is a vectorized match function vmatch that provides the simultaneous substitutions for a vector of patterns in a vector of terms.

$$\mathsf{vmatch}((p_1,\dots,p_n),(t_1,\dots,t_n),C) =$$
$$\{\sigma : \sigma_1 \in \mathsf{match}(p_1,t_1), \sigma_2 \in \mathsf{match}(\sigma_1(p_1),t_2)\circ\sigma_1, \dots, \sigma_n \in \mathsf{match}(\sigma_{n-1}(p_n),t_n)\circ\sigma_{n-1}, \sigma \in \mathsf{check}(C,\sigma_n)\}$$

Sometimes we would like to unbind some variables in a substitution. If $\theta \in \mathrm{VEnv}$ and $Y \subseteq X$ is a subset of variables, θ_{-Y} is the substitution which removes the binding for the variables in Y, i.e.

$$\theta_{-Y}(x) = \begin{cases} x & \text{if } x \in Y \\ \theta(x) & \text{otherwise} \end{cases}$$

The set Y will usually be the set of variables that occur in a term $\mathrm{ocurr}(t)$ or the unbound variables of a matching pattern and condition. We define the latter for $C_i \in \mathit{Condition}$ without assignment fragments as

$$\mathit{unbound}(P, C_1 \wedge l_1 := r_1 \wedge \cdots \wedge C_n \wedge l_n := r_n \wedge C_{n+1}) = \mathrm{ocurr}(P) \cup \mathrm{ocurr}(l_1) \cup \cdots \cup \mathrm{ocurr}(l_n)$$

3 Denotational semantics

The complete definition of the denotational semantics of $\mathrm{Strat}_{\mathcal{R},\Omega}$ is given in Figure 1. Despite not being made explicit, the semantics depend on the named strategies defined in the module. Let m be the size of definitions set $|D|$, and suppose strategy definitions are numbered $(sl_i, \vec{p}_i, \delta_i, C_i) \in D$ from 1 to m. As we will see, the order is irrelevant for the semantics, but it is useful for the presentation. For simplicity, we confuse $sl \in \Omega$ with the plain syntactical label sl that appear in strategy terms. This can be done without loss of generality since overloaded strategy names in an expression must have been resolved before considering semantics.

For each strategy definition $i \in \{1,\dots,m\}$, we consider a semantic function variable d_i. The semantics of any strategy expression can be described as a function $\mathrm{SFun}^m \to \mathrm{SFun}$ on these variables, so we can construct the function $F : \mathrm{SFun}^m \to \mathrm{SFun}^m$, whose least fixed point let us instantiate d_i so that it equals $[\![\delta_i]\!]_\Delta$,

$$F(d_1,\dots,d_m) = ([\![\delta_1]\!]_\Delta,\dots,[\![\delta_m]\!]_\Delta) \qquad (d_1,\dots,d_m) := \mathsf{FIX}\, F$$

Formally, the following two results ensure the correctness of the semantic definitions. Their proofs can be found in Appendix A.2.

▸ **Theorem 1.** Given a rewrite theory $\mathcal{R} = (\Sigma, E \uplus Ax, R)$ with strategies (Ω, D), the functional $F : \mathrm{SFun}^m \to \mathrm{SFun}^m$ given by

$$F(d_1,\dots,d_m) := ([\![\delta_1]\!]_{(d_1,\dots,d_m)},\dots,[\![\delta_m]\!]_{(d_1,\dots,d_m)})$$

is well defined, monotonic and continuous. Then F has a least fixed point

$$\mathsf{FIX}\, F = \sup\{F^n(\{\perp\},\dots,\{\perp\}) : n \in \mathbb{N}\}$$

▸ **Corollary 2.** Given a rewrite theory with strategies as in the previous theorem, the semantics of any strategy term is well defined and $d_i = [\![\delta_i]\!]_\Delta$.

BASE CASES

$$[\![\texttt{idle}]\!]_\Delta(\theta,t)=\{t\} \qquad [\![\texttt{fail}]\!]_\Delta(\theta,t)=\emptyset$$

$$[\![\texttt{match}\ P\ \texttt{s.t}\ C]\!]_\Delta(\theta,t)=\begin{cases}\{t\} & \text{if } \mathsf{mcheck}(P,t,C,\theta)\neq\emptyset\\ \emptyset & \text{otherwise}\end{cases}$$

RECURSIVE CASES

$$[\![\alpha\ ;\ \beta]\!]_\Delta=[\![\beta]\!]_\Delta\circ[\![\alpha]\!]_\Delta \qquad [\![\alpha\,|\,\beta]\!]_\Delta(\theta,t)=[\![\alpha]\!]_\Delta(\theta,t)\cup[\![\beta]\!]_\Delta(\theta,t)$$

$$[\![\alpha^*]\!]_\Delta(\theta,t)=\bigcup_{n=0}^{\infty}[\![\alpha]\!]^n_\Delta(\theta,t)\cup\{\bot : [\![\alpha]\!]^n_\Delta(\theta,t)\neq\emptyset \text{ for all } n\in\mathbb{N}\}$$

$$[\![\alpha\ ?\ \beta\ :\ \gamma\,]\!]_\Delta(\theta,t)=\begin{cases}[\![\gamma]\!]_\Delta(\theta,t) & \text{if } [\![\alpha]\!]_\Delta(\theta,t)=\emptyset\\ [\![\beta]\!]_\Delta\circ[\![\alpha]\!]_\Delta(\theta,t) & \text{otherwise}\end{cases}$$

$$[\![\texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \dots,x_n\ \texttt{using}\ \alpha_n]\!]_\Delta(\theta,t)=$$
$$\bigcup_{\sigma\in\mathsf{mcheck}(P,t,C,\theta)}\ \mathsf{let}\ t_1\leftarrowtail[\![\alpha_1]\!]_\Delta(\sigma,\sigma(x_1)),\dots,t_n\leftarrowtail[\![\alpha_n]\!]_\Delta(\sigma,\sigma(x_n)) : \{\sigma[x_1/t_1,\dots,x_n/t_n](P)\}$$

$$[\![rl\,[x_1\,\texttt{<-}\,t_1,\dots,x_n\,\texttt{<-}\,t_n]\,\{\alpha_1,\dots,\alpha_m\}]\!]_\Delta(\theta,t)=$$
$$\mathsf{ruleApply}(t,rl,\mathrm{id}[x_1/\theta(t_1),\dots,x_n/\theta(t_n)];\alpha_1\cdots\alpha_m,\theta)$$

$$[\![sl(t_1,\dots,t_n)]\!]_\Delta(\theta,t)=f_{sl,\Delta}(\theta(t_1),\dots,\theta(t_n),t)$$

where

$$f_{sl,\Delta}(\vec{s},t)=\bigcup_{(sl,\vec{p}_i,\delta_i,C_i)\in D}\ \bigcup\{d_i(\sigma,t)\,:\,\sigma\in\mathsf{vmatch}(\vec{p}_i,\vec{s},C_i)\}$$

Figure 1: Denotational semantics of the Maude strategy language

3.1 A semantics without scopes

In the first articles describing the Maude strategy language [8, 2, 7], the results of a strategy were described using a set-theoretic semantics without scopes, in which the one in this document is based. However, these early descriptions did not consider variable bindings and formal equations for their semantics were not provided. Still, it is possible to define an equivalent semantics without explicit mention to a variable environment, by applying the substitution directly to the strategy expressions. Some care should be taken with the `matchrew` operators, since their variables naming subterms to be rewritten cannot be vanished. Hence, the definition of substitution application to strategy expressions could not be a straightforward variable substitution.

▸ **Definition.** Given a rewrite theory $\mathcal{R}$ with strategies, the application of a substitution $\theta : X \to T_\Sigma(X)$ to a strategy expressions $\tilde{\theta} : \mathrm{Strat}_{\mathcal{R},\Omega} \to \mathrm{Strat}_{\mathcal{R},\Omega}$ and conditions $\hat{\theta} : \mathit{Condition} \to \mathit{Condition}$ is defined by

1. $\tilde{\theta}(\texttt{fail}) = \texttt{fail}$
2. $\tilde{\theta}(\texttt{idle}) = \texttt{idle}$
3. $\tilde{\theta}(\alpha\ \texttt{;}\ \beta) = \tilde{\theta}(\alpha)\ \texttt{;}\ \tilde{\theta}(\beta)$
4. $\tilde{\theta}(\alpha\,\texttt{|}\,\beta) = \tilde{\theta}(\alpha)\ \texttt{|}\ \tilde{\theta}(\beta)$
5. $\tilde{\theta}(\alpha^*) = \tilde{\theta}(\alpha)^*$
6. $\tilde{\theta}(\alpha\ \texttt{?}\ \beta\ \texttt{:}\ \gamma\,) = \tilde{\theta}(\alpha)\ \texttt{?}\ \tilde{\theta}(\beta)\ \texttt{:}\ \tilde{\theta}(\gamma)\,.$
7. $\tilde{\theta}(\texttt{match}\ P\ \texttt{s.t}\ C) = \texttt{match}\ \theta(P)\ \texttt{s.t}\ \hat{\theta}(C).$
8. $\tilde{\theta}(rl\,\texttt{[}x_1\ \texttt{<-}\ t_1, \dots, x_n\ \texttt{<-}\ t_n\texttt{]}\,\texttt{\{}\alpha_1, \dots, \alpha_m\texttt{\}}) = rl\,\texttt{[}x_1\ \texttt{<-}\ \theta(t_1), \dots, x_n\ \texttt{<-}\ \theta(t_n)\texttt{]}\,\texttt{\{}\tilde{\theta}(\alpha_1), \dots, \tilde{\theta}(\alpha_m)\texttt{\}}$
9. $\tilde{\theta}(\texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \dots, x_n\ \texttt{using}\ \alpha_n) = \texttt{matchrew}\ \theta_{-\{x_1,\dots,x_n\}}(P)$
   $\texttt{s.t.}\ x_{k_1}\ \texttt{=}\ \theta(x_{k_1}) \wedge \cdots \wedge x_{k_m}\ \texttt{=}\ \theta(x_{k_m}) \wedge \hat{\theta}(C)\ \texttt{by}\ x_1\ \texttt{using}\ \tilde{\theta}(\alpha_1),\ \dots, x_n\ \texttt{using}\ \tilde{\theta}(\alpha_n)$
   where $x_{k_1}, \dots, x_{k_m}$ are the subset of variables in $x_1, \dots, x_n$ such that $\theta(x) \neq x$.
   More precisely, these variables should be replaced in the pattern, condition and substrategies by fresh variables disjoint from those which may appear in substitutions, so that subsequent substitutions do not instantiate them with alternative values.
10. $\tilde{\theta}(sl\texttt{(}t_1, \dots, t_n\texttt{)}) = sl\texttt{(}\theta(t_1), \dots, \theta(t_n)\texttt{)}$

and

1. $\hat{\theta}(\texttt{true}) = \texttt{true}.$
2. $\hat{\theta}(l\ \texttt{=}\ r \wedge C) = \theta(l)\ \texttt{=}\ \theta(r) \wedge \hat{\theta}(C).$
3. $\hat{\theta}(l\ \texttt{:=}\ r \wedge C) = \theta(l)\ \texttt{:=}\ \theta(r) \wedge \hat{\theta}(C).$
4. $\hat{\theta}(t\ \texttt{:}\ s \wedge C) = \theta(t)\ \texttt{:}\ s \wedge \hat{\theta}(C).$
5. $\hat{\theta}(l\ \texttt{=>}\ r \wedge C) = \theta(l)\ \texttt{=>}\ \theta(r) \wedge \hat{\theta}(C).$

In the following, we will simply write θ instead of $\tilde{\theta}$ or $\hat{\theta}$, as we already do for the extension of substitutions to terms. Regarding semantics, a natural property must be satisfied: the value of any strategy expression α in a context θ must be the same as the value of $\theta(\alpha)$ in the empty context. It is formally expressed in the following proposition.

▸ **Proposition 3.** Given a rewrite theory with strategies $\mathcal{R}$. For any strategy $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$, substitution $\theta : X \to T_\Sigma(X)$ and term $t \in T_\Sigma(X)$

$$[\![\alpha]\!]_\Delta(\theta, t) = [\![\theta(\alpha)]\!]_\Delta(\mathrm{id}, t)$$

Using this property, the semantics definition could be rewritten so that no variable bindings are passed on. This yields an extension of the old set-theoretic semantics [8, 2, 7], that we can also define in terms of the new semantics, and whose equations we can prove and complete.

▸ **Definition.** Given a rewrite theory with strategies $\mathcal{R} = (\Sigma, E \uplus Ax, R; \Omega, D)$, the *old set-theoretic semantics* is defined as

$$[\![\bullet\ @\ \bullet]\!] : \mathrm{Strat}_{\mathcal{R},\Omega} \times T_\Sigma(X) \to \mathcal{P}_\perp(T_\Sigma(X)) \qquad [\![\alpha\ @\ t]\!] := [\![\alpha]\!](\mathrm{id}, t)$$

▸ **Proposition 4.** Given a rewrite theory with strategies $\mathcal{R} = (\Sigma, E, R; \Omega, D)$, and $\alpha, \beta, \gamma \in \mathrm{Strat}_{\mathcal{R},\Omega}$

1. $[\![\texttt{idle} \mathbin{@} t]\!] = \{t\}$
2. $[\![\texttt{fail} \mathbin{@} t]\!] = \emptyset$
3. $[\![\alpha \,|\, \beta \mathbin{@} t]\!] = [\![\alpha \mathbin{@} t]\!] \cup [\![\beta \mathbin{@} t]\!]$
4. $[\![\alpha \,;\, \beta \mathbin{@} t]\!] = \mathsf{let}\ t' \leftarrowtail [\![\alpha \mathbin{@} t]\!] : [\![\beta \mathbin{@} t']\!]$
5. $[\![\alpha \,?\, \beta : \gamma \mathbin{@} t]\!] = \begin{cases} [\![\alpha \,;\, \beta \mathbin{@} t]\!] & \text{si } [\![\alpha \mathbin{@} t]\!] \neq \emptyset \\ [\![\gamma \mathbin{@} t]\!] & \text{si } [\![\alpha \mathbin{@} t]\!] = \emptyset \end{cases}$
6. $[\![sl(t_1, \ldots, t_n) \mathbin{@} t]\!] = \bigcup_{i=1}^{m_{sl}} \bigcup_{\theta \in \mathsf{vmatch}(sdp_{sl,i}, (t_1, \ldots, t_n), C_{sl,i})} [\![\theta(\delta_{sl,i}) \mathbin{@} t]\!]$.

4 Operational semantics

The denotational semantics in the previous section characterizes the results of the strategy computations. However, we are also interested in the process, in the intermediate steps [10, 12, 11]. The following non-deterministic operational semantics fills that gap, but we have to define a more complex class of intermediate states:

▸ **Definition.** A *context stack* is a word in $\text{Stack} = (\text{Strat}_{\mathcal{R},\Omega} \cup \text{VEnv})^*$. An *execution state* $\mathcal{X}S_{\mathcal{R}}$ is:

1. A pair $t \mathbin{@} s$ composed of a term $t \in T_\Sigma(X)$ and a context stack $s \in \text{Stack}$.
2. An expression of the form
$$\text{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} s$$
where $x_1, \ldots, x_n \in X$, $q_1, \ldots, q_n \in \mathcal{X}S_{\mathcal{R}}$, $t \in T_\Sigma(X)$ and $s \in \text{Stack}$.
3. An expression of the form
$$\text{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \mathbin{@} s$$
where $p, t, r \in T_\Sigma(X)$, $q \in \mathcal{X}S_{\mathcal{R}}$, $\vec{\alpha} \in \text{Strat}^*_{\mathcal{R},\Omega}$, a context c, $\sigma, \theta_s \in \text{VEnv}$, C a rule condition, and $s \in \text{Stack}$.

The stack will be used to store the pending strategies and the nested variable contexts. Item 2 represents parallel `matchrew` executions, and item 3 condition fragment rewriting when applying rules. States of the form $t \mathbin{@} \varepsilon$ are called *solutions*. All these states host information both about the subject term being rewritten and the progress of the strategy execution. In particular, they are uniquely associated to a term:

▸ **Definition.** The *current term* of an execution state $\text{cterm} : \mathcal{X}S_{\mathcal{R}} \to T_\Sigma(X)$ is

1. $\text{cterm}(t \mathbin{@} s) = t$.
2. $\text{cterm}(\text{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} s) = t[x_1/\text{cterm}(q_1), \ldots, x_n/\text{cterm}(q_n)]$.
3. $\text{cterm}(\text{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \mathbin{@} s) = t$.

▸ **Definition.** The *current variable context* of a context stack $\text{vctx} : \text{Stack} \to \text{VEnv}$ is

$$\text{vctx}(\varepsilon) = \text{id} \qquad \text{vctx}(\alpha\, s) = \text{vctx}(s) \qquad \text{vctx}(\sigma\, s) = \sigma$$

for $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $\sigma \in \text{VEnv}$.

Two small-step relations are defined for two distinct levels: system behavior (rewriting with rules) and strategy control (auxiliary tasks to manage strategy execution). In the following, $\to_s$ is the system relation and $\to_c$ the control relation. When the distinction is not necessary, we use the union of both $\to_{s,c} = \to_s \cup \to_c$. Another relation $\twoheadrightarrow = \to_s \circ \to_c^*$ includes a single rule rewrite and all the previous work needed to apply it. This relation is particularly meaningful because whenever $q_1 \twoheadrightarrow q_2 \twoheadrightarrow \ldots \twoheadrightarrow q_n \twoheadrightarrow \ldots$ the rewriting system controlled by the strategy evolves

$$\text{cterm}(q_1) \to_{\mathcal{R}}^1 \text{cterm}(q_2) \to_{\mathcal{R}}^1 \ldots \to_{\mathcal{R}}^1 \text{cterm}(q_n) \to_{\mathcal{R}}^1 \ldots$$

The operational semantics is given in Figures 2 and 3. There, $\alpha, \beta, \gamma \in \text{Strat}_{\mathcal{R},\Omega}$, $s \in \text{Stack}$, C, C' are rule conditions, and C_0 an equational condition. $\theta \in \text{VEnv}$ always refer to $\text{vctx}(s)$. The rules for the `a` and

$$
\begin{array}{rcl}
t \,@\, (\alpha \,;\, \beta)\, s & \to_c & t \,@\, \alpha\beta\, s \\
t \,@\, (\alpha \,|\, \beta)\, s & \to_c & t \,@\, \alpha\, s \\
t \,@\, \alpha^*\, s & \to_c & t \,@\, s \\
t \,@\, (\alpha \;?\; \beta \;:\; \gamma)\, s & \to_c & t \,@\, \alpha\beta\, s
\end{array}
\qquad
\begin{array}{rcl}
t \,@\, \theta\, s & \to_c & t \,@\, s \\
t \,@\, (\alpha \,|\, \beta)\, s & \to_c & t \,@\, \beta\, s \\
t \,@\, \alpha^*\, s & \to_c & t \,@\, \alpha\alpha^*\, s \\
t \,@\, \texttt{idle}\, s & \to_c & t \,@\, s
\end{array}
$$

$$
t \,@\, (\texttt{match}\ P\ \texttt{s.t}\ C)\, s \to_c t \,@\, s \qquad \text{if } \mathsf{mcheck}(P, t, C, \theta) \neq \emptyset
$$

$$
t \,@\, sl(t_1, \ldots, t_n)\, s \to_c t \,@\, \delta_i\, \sigma\, s \qquad \text{if } \sigma \in \mathsf{vmatch}(\vec{p}_i, (\theta(t_1), \ldots, \theta(t_n)), C_i) \text{ and } (sl, \vec{p}_i, \delta_i, C_i) \in D
$$

$$
[\text{else}]\ \frac{}{t \,@\, \alpha \;?\; \beta \;:\; \gamma\ s \to_c t \,@\, \gamma\, s} \text{ if all executions from } t \,@\, \alpha\, \theta \text{ are finite without solutions}
$$

Rewriting of subterms

$$
\begin{array}{l}
t \,@\, (\texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \ldots, x_n\ \texttt{using}\ \alpha_n)\, s \\
\quad \to_c \mathrm{subterm}(x_1 : \sigma(x_1) \,@\, \alpha_1\, \sigma, \ldots, x_n : \sigma(x_n) \,@\, \alpha_n\, \sigma; \sigma_{-\{x_1,\ldots,x_n\}}(P)) \,@\, s \\
\qquad \text{if } \sigma \in \mathsf{mcheck}(P, t, C, \theta)
\end{array}
$$

$$
\mathrm{subterm}(x_1 : t_1 \,@\, \varepsilon, \ldots, x_n : t_n \,@\, \varepsilon; t) \,@\, s \to_c t[x_1/t_1, \ldots, x_n/t_n] \,@\, s
$$

$$
[\mathrm{prl}_c]\ \frac{q_i \to_c q_i'}{\mathrm{subterm}(\ldots, x_i : q_i, \ldots; t) \,@\, s \to_c \mathrm{subterm}(\ldots, x_i : q_i', \ldots; t) \,@\, s}
$$

Rewriting conditions

$$
[\mathrm{rewc}_c]\ \frac{q \to_c q'}{\mathrm{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \,@\, s \to_c \mathrm{rewc}(p : q', \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \,@\, s}
$$

$$
[\mathrm{rewc}_s]\ \frac{q \to_s q'}{\mathrm{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \,@\, s \to_c \mathrm{rewc}(p : q', \sigma, C, \vec{\alpha}, \theta_s, r, c; t) \,@\, s}
$$

$$
\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0 \wedge l \;\texttt{=>}\; p' \wedge C, \alpha\vec{\alpha}, \theta_s, r, c; t) \,@\, s \to_c \mathrm{rewc}(p' : \sigma'(l) \,@\, \alpha\, \theta_s, \sigma', C, \vec{\alpha}, \theta_s, r, c; t) \,@\, s
$$

$$
\text{if } \sigma' \in \mathsf{mcheck}(p, t', C_0, \sigma)
$$

Figure 2: Operational semantics for the Maude strategy language (control)

$$t \,@\, rl\,[x_1 \texttt{<-}\, t_1, \ldots, x_n \texttt{<-}\, t_n]\; s \to_s t' \,@\, s \qquad \text{if } t' \in \mathsf{ruleApply}(t, rl, \mathrm{id}[x_1/\theta(t_1), \ldots, x_n/\theta(t_n)])$$

$$t \,@\, rl\,[x_1 \texttt{<-}\, t_1, \ldots, x_n \texttt{<-}\, t_n]\,\{\alpha_1, \ldots, \alpha_k\}\; s \to_c \mathrm{rewc}(p : \sigma(t_0) \,@\, \alpha_1\theta_s, \sigma, C', \alpha_2 \cdots \alpha_k, \theta, r, c; t) \,@\, s$$
$$\text{if } (rl, l, r, C) \in R, \mathrm{nrewf}(C) = k, C = C_0 \wedge t_0 \texttt{ => } p \wedge C', \theta = \mathrm{vctx}(s),$$
$$\text{and } (\sigma, c) \in \mathsf{amcheck}(l, t, C_0, \mathrm{id}[x_1/\theta(t_1), \ldots, x_n/\theta(t_n)])$$

$$[\mathrm{prl}_s]\ \frac{q_i \to_s q_i'}{\mathrm{subterm}(\ldots, x_i : q_i, \ldots; t) \,@\, s \to_s \mathrm{subterm}(\ldots, x_i : q_i', \ldots; t) \,@\, s}$$

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0, \vec{\alpha}, r, c; t) \,@\, s \to_s c(\sigma'(r)) \,@\, s$$
$$\text{if } \sigma' \in \mathsf{mcheck}(p, t', C_0, \sigma)$$

Figure 3: Operational semantics for the Maude strategy language (system)

`x` variants of `match` and `matchrew` are omitted as they are too similar. Only note that in the first `matchrew` rule, the context c must be applied to the last entry of the subterm state, $c(\sigma_{-\{x_1,\ldots,x_n\}}(P))$.

Both $\to_c$ and $\to_s$ are not deterministic: they may decide between different alternatives and loose solutions on each step. Executions can be seen in different ways: *executions* or *execution sequences* are finite or infinite sequences of states like

$$t \,@\, (rl_1 \,|\, rl_2)\; s \to_c t \,@\, rl_1\; s \to_s t_1 \,@\, s$$

Executions may get stuck or arrive to a solution $t \,@\, \varepsilon$. They consider only one of the possible successors. When all of them are considered, we refer to an *execution tree*

$$t \,@\, (rl_1 \,|\, rl_2)\; s \;\begin{cases} \xrightarrow{\mathrm{c}}\; t \,@\, rl_1\; s \xrightarrow{\mathrm{s}} t_1 \,@\, s \\ \xrightarrow{\mathrm{c}}\; t \,@\, rl_2\; s \xrightarrow{\mathrm{s}} t_2 \,@\, s \end{cases}$$

But there is another tree in the picture: the *proof tree* for a single step of the semantics. For example, when rewriting subterms

$$\frac{t \,@\, (rl_1 \,|\, rl_2)\; s \to_c t \,@\, rl_1}{\mathrm{subterm}(x_1 : t \,@\, (rl_1 \,|\, rl_2)\; s; t) \,@\, s' \to_c \mathrm{subterm}(x_1 : t \,@\, rl_1\; s; t) \,@\, s'}\ \text{using } [\mathrm{prl}_c]$$

The condition of the [else] rule refers to the execution tree from $t \,@\, \alpha\,\theta$. It is the translation of $[\![\alpha]\!](\theta, t) = \emptyset$ for the denotational semantics. Note that the evaluation of this condition may not finish. Thus, $\to_c$ is undecidable and so $\to_{s,c}$ and $\twoheadrightarrow$ are.

While derivations are linear, proofs almost never are. The main rules are usually applied to substates by using $[\mathrm{prl}_c]$, $[\mathrm{prl}_s]$, $[\mathrm{rewc}_c]$ or $[\mathrm{rewc}_s]$. In the case of [else], all derivations from the condition must be proved to finish and fail. However, a practical execution of these rules is easier. For example, the information about the execution of $t \,@\, \alpha$ in the tentative execution of the positive branch by $t \,@\, \alpha \texttt{ ? } \beta : \gamma \to_c t \,@\, \alpha\beta$ can be used to trigger the [else] rule if all attempts from $t \,@\, \alpha$ have failed.

4.1 Some properties

An important property of the new semantics is that it is equivalent to the previous denotational semantics in the following way:

▸ **Theorem 1.** For all $t, t' \in T_\Sigma(X)$, $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$ and $\theta \in \mathrm{VEnv}$

$$t' \in [\![\alpha]\!](\theta, t) \quad \iff \quad t \mathbin{@} \alpha\,\theta \to^*_{s,c} t' \mathbin{@} \varepsilon$$

and $\bot \in [\![\alpha]\!](\theta, t)$ iff there is an infinite derivation from $t \mathbin{@} \alpha\,\theta$.

The novelty of the operational semantics, apart from the details about the intermediate states of the rewriting process, is that non-termination is not only a binary condition, but provides valuable information in the form of infinite execution sequences.

▸ **Definition.** For any $q_0 \in \mathcal{XS}_\mathcal{R}$

1. The *reachable states* from q_0 are $\{q : q_0 \to^*_{s,c} q\}$.
2. The *reachable terms* from q_0 are $\cup_{q : q_0 \to^*_{s,c} q} \mathrm{terms}(q)$ where

$$\mathrm{terms}(q) = \mathrm{cterm}(q) \cup \begin{cases} \mathrm{terms}(q_1) \cup \cdots \cup \mathrm{terms}(q_n) & \text{if } q = \mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \\ \mathrm{terms}(q') & \text{if } q = \mathrm{rewc}(x : q', \ldots) \\ \{\mathrm{vctx}(s)(t_1), \ldots, \mathrm{vctx}(s)(t_n)\} & \text{if } q = t \mathbin{@} sl(t_1, \ldots, t_n)\, s \end{cases}$$

▸ **Proposition 2.** For any $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$ and $t \in T_{\Sigma/E}$, the set of reachable states from $t \mathbin{@} \alpha$ is finite if any of the following condition holds:

1. α does not contain iterations or recursive function calls.
2. The reachable terms from $t \mathbin{@} \alpha$, in rewriting conditions, and subterm rewritings are finitely many; strategy call arguments only takes a finite number of values; and all recusive calls in strategy definitions are tail.

   We must also consider that variable environments are replaced instead of pushed to the stack when other substitution is on top. This *optimization* does not affect the semantics.

The following definitions will be used to fix the concept of *abstract strategy* [5, 4] for any strategy expression α, as a language of all allowed execution traces. Strategies are used both to express computations and to describe the behavior of reactive systems, for which finite length and infinite words are respectively relevant, so neither of them should be omitted. Moreover, not all finite executions are interesting: some lead to a failure or deadlock state, and do not correspond to solutions for the denotational semantics.

▸ **Definition.**

1. The language of *infinite executions* from q_0 is

$$\mathrm{Ex}_\omega(q_0) = \{q_0\, q_1 \cdots q_n \cdots \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots\}$$

2. The language of *finite (complete) executions* from q_0 is

$$\mathrm{Ex}_{\mathrm{fin}}(q_0) = \{q_0\, q_1 \cdots q_n \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \wedge q_n \text{ is an end }\}$$

   and q is an end if $q \to^*_{s,c} \mathrm{cterm}(q) \mathbin{@} \varepsilon$ or $q \twoheadrightarrow q'$ never holds for $q' \in \mathcal{XS}_\mathcal{R}$.

3. The language of *successful executions* from q_0 is

$$\mathrm{Ex}_{\mathrm{succ}}(q_0) = \{q_0 \cdots q_n \mid q_0 \twoheadrightarrow \cdots \twoheadrightarrow q_n \to^*_c \mathrm{cterm}(q_n) \mathbin{@} \varepsilon\}$$

Execution states are only an artifice to maintain the required control data for the execution of a strategy. We are interested in the evolution of the underlying rewriting system, in terms of $T_{\Sigma/E}$ and $\to^1_\mathcal{R}$. Abusing of notation, a function $f : X \to Y$ may be extended to $f : X^* \to Y^*$ by $f(x_1 \cdots x_n) \coloneqq f(x_1) \cdots f(x_n)$, and to $f : X^\omega \to Y^\omega$ similarly. The application of f to a subset of X is standard, the image of a set by a function.

▸ **Definition.**

1. The language of *infinite execution traces* from q_0 is $T_\omega(q_0) = \text{cterm}(\text{Ex}_\omega(q_0))$.
2. The language of *finite execution traces* from q_0 is $T_{\text{fin}}(q_0) = \text{cterm}(\text{Ex}_{\text{fin}}(q_0))$.
3. The language of *successful execution traces* from q_0 is $T_{\text{succ}}(q_0) = \text{cterm}(\text{Ex}_{\text{succ}}(q_0))$.

▸ **Definition.** For any $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ the *abstract strategy* α from states in I is

$$E(\alpha, I) = \bigcup_{t \in I} T_\omega(t \mathbin{@} \alpha) \cup T_{\text{succ}}(t \mathbin{@} \alpha)$$

For some applications, it is convenient to consider strategies as pure ω-languages. This can be done completing $\twoheadrightarrow$ to be total with $q \twoheadrightarrow \text{cterm}(q) \mathbin{@} \varepsilon$ if $q \to_c^* \text{cterm}(q) \mathbin{@} \varepsilon$, for all state q. In this case the definition is as simple as

$$E(\alpha, I) = \{\text{cterm}(q_0)\,\text{cterm}(q_1) \cdots \text{cterm}(q_n) \cdots \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots\}$$

Equivalently, successful traces can be extended by repeating the last state forever. The next propositions relate regular languages and reachable states cardinality:

▸ **Proposition 3.** For any $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ finite, if the reachable states from $t \mathbin{@} \alpha$ are finite for $t \in I$, the extended abstract strategy $E(\alpha, I)$ is an ω-regular language, and a Büchi automaton for $E(\alpha, I)$ is $\mathcal{A} = (Q, \text{cterm}(Q), \delta, \{\mathit{start}\}, Q)$ where $Q = \{\mathit{start}\} \cup \{q \in \mathcal{XS}_\mathcal{R} \mid t \in I \ \wedge\ t \mathbin{@} \alpha \twoheadrightarrow^* q\}$ and

$$\begin{aligned}
\delta(\mathit{start}, t) &= \{\ t \mathbin{@} \alpha\ \} && \text{if } t \in I \\
\delta(\mathit{start}, t) &= \emptyset && \text{if } t \notin I \\
\delta(q, t) &= \{\ q' : q \twoheadrightarrow q' \ \wedge\ \text{cterm}(q') = t\ \} && \text{for } q \in Q \backslash \{\mathit{start}\} \\
&\quad \cup \{\ t \mathbin{@} \varepsilon : \text{if } q \to_c^* t \mathbin{@} \varepsilon\ \}
\end{aligned}$$

Nevertheless, the reciprocal does not hold, the language $E(\alpha, I)$ can be ω-regular and the reachable states from α be infinitely many. In a strategy module with `sd st(X) := fail | st(s(X))` consider $\alpha :=$ `st(0)`. No matter the rewriting theory, $t \mathbin{@} \alpha$ cannot be continued by $\twoheadrightarrow$. So $E(\alpha, I) = \emptyset$, which is ω-regular. But infinite states of the form $t \mathbin{@} \texttt{fail}[x \mapsto N] \ldots [x \mapsto 1][x \mapsto 0]$ are reachable from the origin.

Moreover, the automaton $\mathcal{A}$ for $E(\alpha, I)$ has always trivial Büchi conditions. Hence, abstract strategies whose strategy expression generator is executed in a finite number of states are a restricted subclass of ω-regular languages. In the following section, an alternative abstract strategy definition will be proposed, such that they cover the whole space of ω-regular languages.

4.2 Another definition for strategies

The interpretation of α as an abstract strategy has some downsides related to infinite traces. For example,

- The semantics of regular expressions, included in the language, is not completely respected, because iterations can be executed infinitely, against the usual meaning of the Kleene star.
- Subterm states with failed substrategies can appear in infinite traces.

For instance, being `id` an identity rule, `matchrew f(X, Y) by X using id*, Y using fail` can lead to an infinite execution. The second problem is not much interesting and relatively easy to solve, so we will focus on the first problem. Avoiding infinite iteration will require more attention and computational cost.

For any state $q \in \mathcal{XS}_\mathcal{R}$, a position $p \in \mathbb{N}^*$ allows selecting a substate $q|_p$ inside q. The empty position ε refers to the whole state q, and if q is a subterm state with n substrategies, 0 to $n-1$ point to the substates that rewrite against them. Nested positions are combined by juxtaposition, the outer index first.

▸ **Definition.** Given an execution $q_0 \cdots q_n \cdots \in \mathrm{Ex}_\omega(q_0)$, we define

1. The states where an iteration of α starts at position p and with base stack s,

$$\mathrm{Iter}(p, \alpha, s) = \{\ k \in \mathbb{N} : q_k|_p \to_c^* t_k \mathbin{@} \alpha^* s,\ q_{k+1}|_p = t_{k+1} \mathbin{@} s'\alpha^* s,\ t_k \mathbin{@} \alpha\, \mathrm{vctx}(s) \twoheadrightarrow t_{k+1} \mathbin{@} s'\theta\ \}$$

2. The states where an iteration of α ends at position p with base stack s,

$$\mathrm{Leave}(p, \alpha, s) = \{\ k \in \mathbb{N} : q_k|_p \to_c^* t_k \mathbin{@} \alpha^* s,\ t_k \mathbin{@} s \twoheadrightarrow q_{k+1}|_p\ \}$$

3. The consecutive iterations from the state n at position p,

$$\mathrm{ConsIter}(n, p) = \begin{cases} \{\ k \in \mathrm{Iter}(p, \alpha, s) : \forall j\ \ n \le j \le k \quad j \notin \mathrm{Leave}(p, \alpha, s)\ \} & \text{if } q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s \\ \emptyset & \text{otherwise} \end{cases}$$

Leave and ConsIter are introduced to be more specific and only purge infinite consecutive iterations. There is no problem if an iteration $\alpha^* s$ is initiated infinitely many times but it iterates only finitely each. However, this cannot happen, as we will later prove, unless the optimization of Proposition 2 is applied.

▸ **Definition.** An infinite execution $\pi \in \mathrm{Ex}_\omega(q_0)$ *iterates finitely* if for all $n \in \mathbb{N}$ and position p inside q_n, $\mathrm{ConsIter}(n, p)$ is finite.

▸ **Proposition 4.** Given $\pi = q_0 \cdots q_n \cdots \in \mathrm{Ex}_\omega(q_0)$, the following conditions are equivalent:

1. For all $n \in \mathbb{N}$ and position p in q_n such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, $\mathrm{Iter}(p, \alpha, s)$ is finite.
2. π iterates finitely.
3. For any $n \in \mathbb{N}$ and position p in q_n such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, $\mathrm{Iter}(p, \alpha, s)$ is finite or $\mathrm{Leave}(p, \alpha s)$ is infinite.

And the last two conditions are equivalent even with the optimization of Proposition 2.

When only finite iterations are allowed, the only available resource to produce infinite executions is non-terminating recursive strategies. Moreover, any infinite execution iterating finitely visits an infinite number of execution states, unless the optimization is applied. Likewise, it makes an infinite number of strategy calls, as stated in the following proposition where cdepth refers to the number of nested call contexts in a state, formally defined in the appendices.

▸ **Proposition.** Given an execution $\pi \in \mathrm{Ex}_\omega(q_0)$ that iterates finitely, $\mathrm{cdepth}(\pi_n) \to \infty$ when $n \to \infty$.

▸ **Definition.** For any $q_0 \in \mathrm{Strat}_{\mathcal{R},\Omega}$ the set of infinite executions (new definition) is

$$\mathrm{Ex}'_\omega(q) = \{\ \pi \in \mathrm{Ex}_\omega(q) : \pi \text{ iterates finitely}\ \}$$

and the set of infinite traces is $T'_\omega(q_0) = \mathrm{cterm}(\mathrm{Ex}'_\omega(q_0))$.

▸ **Definition.** For any $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ the *abstract strategy* (new definition) α from states in I is

$$E(\alpha, I) = \bigcup_{t \in I} T'_\omega(t \mathbin{@} \alpha) \cup T_{\mathrm{succ}}(t \mathbin{@} \alpha)$$

For the rest of the section, we will study how to construct an ω-regular automaton for $E(\alpha, I)$ when the reachable states are finitely many. Using the third condition of Proposition 4, we will provide some acceptance condition, so that the automaton proposed for the previous definition can be used for the new one. *Streett acceptance conditions* are pairs $\{(A_1, B_1), \ldots, (A_k, B_k)\} \subseteq q^2$ where q is the set of states, and a run π is accepted if $\inf(\pi) \cap A_i = \emptyset$ or $\inf(\pi) \cap B_i \neq \emptyset$ for all $1 \le i \le k$. In our case, the A_i will be some $\mathrm{Iter}(p, \alpha, s)$ and B_i will be $\mathrm{Leave}(p, \alpha, s)$.

▸ **Proposition 5.** For any $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ finite, if the reachable states from $t \mathbin{@} \alpha$ are finitely many for $t \in I$, the extended abstract strategy $E(\alpha, I)$ is an ω-regular language, and a Streett automaton for it is $\mathcal{A} = (Q^2, \mathrm{cterm}(Q \setminus \{start\}), \delta, \{start\} \times I, S)$ where

$$Q = \{start\} \cup \{q \in \mathcal{X}S_{\mathcal{R}} \mid t \in I \ \wedge \ t \mathbin{@} \alpha \twoheadrightarrow^* q\},$$

the Streett conditions are $S = \{\ (A_{p,\alpha,s}, B_{p,\alpha,s}) : (p, \alpha, s) \in J\ \}$ for

$$\begin{aligned} J &= \{\ (p, \alpha, s) : \exists\, q \in Q, t \in T_{\Sigma/E} \quad q|_p \to_c^* t \mathbin{@} \alpha^* s\ \} \\ A_{p,\alpha,s} &= \{\ (q, q') \in Q^2 : q|_p \to_{s,c}^* t \mathbin{@} \alpha\alpha^* s \twoheadrightarrow q'|_p,\ t \in T_{\Sigma/E}\ \} \\ B_{p,\alpha,s} &= \{\ (q, q') \in Q^2 : q|_p \to_{s,c}^* t \mathbin{@} s \twoheadrightarrow q'|_p,\ t \in T_{\Sigma/E}\ \}, \end{aligned}$$

and

$$\delta((q, q'), t) = \{\ (q', q'') \in Q^2 : q' \twoheadrightarrow q'' \ \wedge \ \mathrm{cterm}(q') = t\ \} \cup \{\ (q', t \mathbin{@} \varepsilon) : \text{if } q' \to_c^* t \mathbin{@} \varepsilon\ \}$$

This automaton is valid both with or without the optimization of Proposition 2.

While the states of the automaton are defined as pairs of execution states, the size of the automaton does not need to grow quadratically in practice. Since it is possible to arrive to the same execution state by transitions that enter or leave different iterations, the first state of the tuple is convenient to disambiguate these situations. However, this is only possible with subterm states, and the first entry can be safely removed otherwise. Moreover, Streett automata can be translated to Büchi automata, but paying a blow up of $n(1 + k2^k)$ in the worst case, where n is the number of states, and k the automaton index, i.e. the number of Streett conditions, which coincides with the number of reachable iteration expressions. Hence, reducing k or imposing simpler conditions is convenient.

- If the optimization of Proposition 2 is disabled, the first statement of Proposition 4 can be used instead of the third (or as a particular case of the third) to define the acceptance conditions. Thus, $E(\alpha, I)$ is accepted by a co-Büchi automaton whose acceptance set is the union of all $A_{p,\alpha,s}$.
- Some iterations will not iterate forever due to its own semantics. This is the usual case when iterations are used to reduce something until it is not further possible. So, we only have to impose Streett (or co-Büchi) conditions to those iterations in the automata that generate cycles.
- The effect of optimization is only observed *inside* tail function calls. For all iterations whose base stack s has not been optimized, B can be $\emptyset$, i.e. a co-Büchi condition is enough.

As we have seen, the are many situations where we can simplify, reduce, and even omit the acceptance conditions. However, acceptance condition are essential in some cases, as the following proposition confesses.

As we have seen in the previous section, if $E(\alpha, I)$ is an ω-language, the reachable states from $t_0 \mathbin{@} \alpha$ do not need to be a finite set, for any $t_0 \in I$. But, in this section, we can find a substitute β such that $E(\beta, I) = E(\alpha, I)$ and the reachable states are finitely many.

▸ **Proposition 6.** If $S \subseteq T_\Sigma(X)$ finite and $L \subseteq S^\omega$ is a regular language and $w_i \to_{\mathcal{R}}^1 w_{i+1}$ for all $w \in L$, there is a strategy α and a set $I \subseteq T_\Sigma(X)$ such that $E(\alpha, I) = L$ and the reachable states from $t \mathbin{@} \alpha$ with optimization are finitely many for any $t \in T_\Sigma(X)$.

Reaching an infinite number of states cannot be always avoided by choosing a convenient strategy, since the strategy language is able to express any recursive enumerable abstract strategy. An easy example of non-regular language is that of the paired parenthesis. For a sort with three symbols `(`, `)` and `z`, we can define a module like this:

```
rl [open] : X => ( .
rl [close] : X => ) .
rl [rest] : X => z .

strat paren : Nat @ Symbols .
sd paren(0) := open ; paren(1) | rest .
sd paren(s(N)) := close ; paren(N) | open ; paren(N + 2) .
```

And then the strategy `paren(0)` is not ω-regular.

5 Rewriting semantics

This section presents an updated and improved version of a rewriting-based semantics [7] that transforms a pair (M, SM), i.e. a system module M along with a strategy module that defines strategies for M, into a rewrite theory $\mathcal{S}(M, SM)$ where strategy expressions can be written and applied to terms. The transformed module implements the syntax of the strategy language and the infrastructure and rules to apply them. For this purpose, some function definitions and rules should be added to the transformed module for each strategy construct. First of all, the signature of the transformed module should include some auxiliary infrastructure for substitutions and matching. Declarations with a type annotation like S are generated for each sort in M.

```
sort Substitution .

op _<-_ : VarS S -> Substitution [ctor] .
op none : -> Substitution [ctor] .
op _,_ : Substitution Substitution -> Substitution [ctor assoc id: none] .

op _·_ : S Substitution -> S .
```

A substitution is defined as a list of variable-to-term bindings, and an infix dot operator represents the application of a substitution to a term.

```
sorts Match MatchSet .
subsort Match < MatchSet .
op <_,_> : Substitution S -> Match [ctor] .
op none : -> MatchSet [ctor] .
op __ : MatchSet MatchSet -> MatchSet [ctor assoc comm id: none] .

op [] : -> S [ctor] .

op getMatch : S S EqCondition -> MatchSet .
op getAmatch : S S' EqCondition -> MatchSet .
op getXmatch : S S EqCondition -> MatchSet .
```

Each of the last three operators returns all matches of its second argument into the first, respectively on top, anywhere, or on top with extension. A match is described by a pair containing a substitution and a context. The context *hole* is indicated by means of an overloaded constant `[]`.

```
sorts Condition EqCondition .
subsort EqCondition < Condition .

op trueC : -> EqCondition  [ctor] .
op _=_ : S S -> EqCondition [ctor] .
op _:=_ : S S -> EqCondition [ctor] .
op _: S : S -> EqCondition [ctor] .
op _=>_ : S S -> Condition [ctor] .

op _/\_ : Condition Condition -> Condition [ctor assoc id: trueC] .
op _/\_ : EqCondition EqCondition -> EqCondition [ditto] .

op _·_ : Condition Substitution -> Condition .
```

Equational and rule conditions are defined with the usual syntax, and a substitution can also be recursively applied to them.

Second, strategy language constructs are expressed as Maude operators. Its signature is similar to the meta-representation of strategies in [1, §17.3] but applied to the object level. Hence, we only include some as an example.

```
sorts RuleApp Strat StratCall .
subsorts RuleApp StratCall < Strat < StratList .

op _[_]{_} : Label Substitution StratList -> RuleApp [ctor] .
op match_s.t._ : S EqCondition -> Strat [ctor] .
op _;_ : Strat Strat -> Strat [ctor] .
*** and more

op _·_ : Strat Substitution -> Strat .
```

Substitutions can be applied to strategy expressions too. The equational definition is straightforward except for the case of the `matchrew`. Pattern variables that designate subterms to be rewritten cannot be replaced syntactically because the reference will be lost. However, the conflicting substitution assignment can be translated into an equality condition fragment to be added to the strategy expression. In order to avoid problems with successive substitutions, we should use fresh variables for these new condition fragments, albeit this situation does not happen in practice.

```
ceq matchrew(P:S, C, VSL) · Sb =
        matchrew(P:S · SSb, SCond /\ C · Sb, VSL · Sb)
 if { SSb ; SCond } := splitSubs(Sb, VSL) .

eq splitSubs(X:S <- T:S ; Sb, X:S using E) = { Sb ; X:S = T:S } .
eq splitSubs(Sb, X:S using E) = { Sb ; nil } [owise] .
ceq splitSubs(X:S <- T:S ; Sb, (X:S using E, VSL)) =
        { Sb' ; C /\ X:S = T:S) if { Sb' ; C } := splitSubs(Sb, VSL) .
eq splitSubs(Sb, (X:S using E, VSL)) = splitSubs(Sb, VSL) [owise} .
```

Third, the strategy execution infrastructure is based on a series of tasks and continuations.

```
sorts Task Tasks Cont .
subsort Task < Tasks .
op none : -> Tasks [ctor] .
op __ : Tasks Tasks -> Tasks [ctor assoc comm id: none] .
eq T:Task T:Task = T:Task .

op <_@_> : Strat S -> Task [ctor] .
op sol :  S -> Task [ctor] .
op <_;_> : Tasks Cont -> Task [ctor] .

op chkrw : Condition StratList S S -> Cont [ctor] .
op seq : Strat -> Cont [ctor] .
op ifc : Strat Strat S -> Cont [ctor] .
op mrew : S S' Substitution VarStratList -> Cont [ctor] .
op onec : -> Cont [ctor] .
```

The application of a strategy α to a term t is represented by a task `< `α` @ `t` >`, and solutions are captured in `sol(`t`)` tasks. Tasks can be rewritten and fork new tasks, which represent different search states. They are all gathered in an associative and commutative soap of sort `Tasks`. Nested searches are represented by the `<_;_>` constructor, which additionally contains a *continuation* that the results from the inner search must execute to be a solution for the outer execution level. Continuations are specified as terms of sort `Cont`.

Idle and fail. The `idle` and `fail` meaning is given by the following rules that convert the `idle` task to a solution, and remove the `fail` task.

```
rl < idle @ T:S > => sol(T:S) .
rl < fail @ T:S > => none .
```

Rule application. For each unconditional rule or each conditional rule without rewriting fragments l `=>` r with label *label* in M, a rule as below is appended to the transformed module:

```
crl < label[Sb]{empty} @ T:S > => gen-sols(MAT, r · Sb)
if MAT := getAmatch(l · Sb, T:S, C) .

eq gen-sols(none, T:S') = none .
eq gen-sols(< Sb, Cx:S > MAT, T:S') =
        sol(replace(Cx:S, T:S' · Sb)) gen-sols(MAT, T:S') .
```

Both rule sides are applied the substitution `Sb` from the application expression, which may be empty. Then the partially instantiated lefthand side is matched against the subject term, and their matches are passed to the `gen-sols` function. This function traverses the set generating a solution task for each match, by instantiating the righthand side of the rule with the matching substitution, and building up the context with `replace`.

The treatment of rewriting conditions is much more involved because they must be rewritten according to the given strategies. To handle this situation, we make use of a continuation. Consider a rule

```
rl [label] : l => r if C0 /\ u1 => v1 /\ C1 /\ ... /\ Cn-1 /\ un => vn /\ Cn .
```

where C_k are equational conditions, which may be empty. Let RC be the condition fragments from C_1 to C_n. For each such rule, we generate

```
var C  : EqCondition .
var RC : Condition .

crl < label[Sb]{E1, ..., En} @ T:S >
   => gen-rw-tks(MAT, u1 · Sb, (u1 => v1 /\ RC) · Sb,
                 E1 ... En, r · Sb)
   if MAT := getAmatch(l · Sb, T:S, C0) .

eq gen-rw-tks(none, U:S', RC, EL, Rhs:S'') = none .
eq gen-rw-tks(< Sb, Cx:S > MAT, T:S', RC, (E, EL), Rhs:S'') =
        < < E @ T:S' · Sb > ; chkrw(RC · Sb, (E, EL), Rhs:S'' · Sb, Cx:S) >
        gen-rw-tks(MAT, T:S', RC, (E, EL), Rhs:S'') .
```

The function `gen-rw-tks` traverses the set of matches like `gen-sols`, but a continuation task is generated for each match. Its nested computation applies the first given strategy to the lefthand side of the first rewriting fragment instantiated by the matching substitution. Its `chkrw` continuation stores the condition, the pending controlling strategies, the rule righthand side and the subterm context. This allows checking the condition recursively and stepwise. When a solution is obtained in the nested computation, it must be matched against the fragment righthand side and all the matches must be continued as potentially different condition solutions.

```
crl < sol(R:S) TS ;
      chkrw(U:S => V:S /\ C /\ U':S' => V':S' /\ RC,
            (E, E', EL), Rhs:S'', Cx:S'') >
 => < TS ; chkrw(U:S => V:S /\ C /\ U':S' => V':S' /\ RC,
                 (E, E', EL), Rhs:S'', Cx:S''') >
      gen-rw-tks2(MAT, U':S', (U':S' => V':S' /\ RC),
                  (E', EL), Rhs:S'', Cx:S''')
   if MAT := getMatch(V:S, R:S, C) .

eq gen-rw-tks2(none, T:S', RC, EL, Rhs:S'', Cx:S''') = none .
eq gen-rw-tks2(< Sb, Cx:S > MAT, T:S', RC, (E, EL), Rhs:S'', Cx:S''') =
     < < E @ T:S' · Sb > ;
       chkrw(RC · Sb, (E, EL), Rhs:S'' · Sb, Cx:S''') >
     gen-rw-tks2(MAT, T:S', RC, (E, EL), Rhs:S'', Cx:S''') .
```

Here, `gen-rw-tks2` walks over the matches for the righthand side of the previous condition fragment, and generates continuation tasks that evaluate the next condition fragment as already done for the initial one. Clearly, the base case of this process is reached when no rewriting fragment remains.

```
crl < sol(R:S) TS ; chkrw(U:S => V:S /\ C, E, Rhs:S', Cx:S'') >
 => < TS ; chkrw(U:S => V:S /\ C, E, Rhs:S', Cx:S'') >
    gen-sols2(MAT, Rhs:S', Cx:S'')
 if MAT := getMatch(V:S, R:S, C) .

eq gen-sols2(none, Rhs:S, Cx:S') = none .
eq gen-sols2(< Sb, Cx':S'' > MAT, Rhs:S, Cx:S')
 = sol(replace(Cx:S', Rhs:S · Sb)) gen-sols2(MAT, Rhs:S, Cx:S') .
```

The function `gen-sols2` finally composes the solutions of the rule application by rebuilding the term using the successively instantiated righthand side of the rule. In case any of the nested strategy evaluations fails, the whole rule application fails.

```
rl < none ; chkrw(RC, EL, Rhs:S', Cx:S) > => none .
```

Tests. As described before, tests behave like `idle` if there is a match satisfying the condition, and like a `fail` if there is none.

```
crl < match P:S s.t. C @ T:S > => sol(T:S)
 if < Sb, Cx:S > MAT := getMatch(P, T:S, C) .
```

```
crl < match P:S s.t. C @ T:S > => none
 if getMatch(P:S, T:S, C) = none .
```

The `amatch` and `xmatch` variants are defined by similar pairs of rules. The only difference is the search function, `getAmatch` and `getXmatch`, which can be implemented in Maude by means of the family of `metaMatch` functions.

Regular expressions. Regular expressions can be handled by a series of simple rules:

```
rl < E | E' @ T:S > => < E @ T:S > < E' @ T:S > .
rl < E ; E' @ T:S > => < < E @ T:S > ; seq(E') > .
rl < sol(R:S) TS ; seq(E') > => < E' @ R:S > < TS ; seq(E') > .
rl < none ; seq(E') > => none .
rl < E * @ T:S > => sol(T:S) < E ; (E *) @ T:S > .
eq E + = E ; E *  .
```

The rule for alternation splits the task in two that continue each with one of the alternatives. The concatenation creates a nested task to evaluate the first of the concatenated strategies and leaves the second strategy pending using the `seq` continuation. Each solution found in the nested search is then continued using the strategy in this continuation. When the subsearch runs out of tasks, the task is discarded. The iteration rule, following its recursive definition, produces both a solution for the empty iteration, and a task that evaluates the iteration body concatenated with the iteration itself. The non-empty iteration is equationally reduced to this equivalent expression.

Conditionals. The semantics of the conditional is also expressed by a continuation and a subsearch for the strategy condition. The `ifc` continuation maintains the strategies for both branches of the conditional and the initial term, which will be used if the negative branch has to be evaluated.

```
rl < E ? E' : E'' @ T:S > => < < E @ T:S > ; ifc(E', E'', T:S) > .
rl < sol(R:S) TS ; ifc(E', E'', T:S) > => < E' @ R:S > < TS ; seq(E') > .
rl < none ; ifc(E', E'', T:S) > => < E'' @ T:S > .
```

When a solution is found for the condition, the result is given a task to be continued by the positive branch strategy. Moreover, the conditional `ifc` continuation is transformed in a `seq` continuation, since the execution of the negative branch is already discarded. On the contrary, if the tasks in the subcomputation get exhausted, the negative branch is evaluated in the initial term by means of a new task.

The semantics of the derived operators is implicitly given by equationally translating them to their equivalent expressions:

```
eq E or-else E' = E ? idle : E' .
eq not(E) = E ? fail : idle .
eq try(E) = E ? idle : idle .
eq test(E) = not(not(E)) .
```

Rewriting of subterms. The rewriting of subterms operator rewrites the matched subterms by the given strategies. Like for rewriting conditions, this is handled using a continuation `mrew(P, Sb, Cx, X, VSL)` that holds the main pattern `P`, the substitution `Sb` and the context `Cx` where it occurs in the subject term, the variable whose subterm is currently being rewritten, and the list of pending term-using-strat pairs.

```
crl < amatchrew(P:S, C, VSL) @ T:S0 > => gen-mrew(MAT, P:S, VSL)
 if MAT := getAmatch(P:S, T:S0, C) .

eq gen-mrew(none, P:S, VSL) = none .
ceq gen-mrew(< Sb, Cx:S0 > MAT, P:S, VSL) =
    < < E · Sb @ X:S' · Sb > ; mrew(P:S, Cx:S0, Sb, VSL) >
    gen-mrew(MAT, P:S, VSL)
 if X:S' using E := firstPair(VSL) .
```

For every match of the main pattern in the subject term, a continuation task is created and it starts to evaluate the first strategy in the matched subterm, which is recovered by `X:`S' `· Sb`. The strategy is also applied the substitution `Sb` since it is allowed to contain free occurrences of the matching variables.

When the evaluation of a subterm gives a solution, the `mrew` task is split into two: the first one keeps looking for other solutions for the same subterm, and another one continues with the evaluation of the next subterm. The creation of the last task is similar to the initial case, but the information is obtained from the continuation instead. The result of the subterm rewriting is substituted in the copy of the main pattern carried by the continuation. Like this, when all the subterms have been processed, the copy of the pattern will have the initial subterms replaced by some results, so that the rest of the variables can be instantiated with the initial substitution, and the initial term rebuilt by means of the context stored in the continuation.

```
crl < sol(T:S') TS ; mrew(P:S, Cx:S0, Sb, (X:S' using E', VSL)) > =>
   < TS ; mrew(P:S, Cx:S0, Sb, (X:S' using E', VSL)) >
   < < E · Sb @ Y:S'' · Sb > ; mrew(P:S · (X:S' <- T:S'), Cx:S0, Sb, VSL) >
 if Y:S'' using E := firstPair(VSL) .

rl < sol(T:S') TS ; mrew(P:S, Cx:S0, Sb, X:S' using E) > =>
   < TS ; mrew(P:S, Cx:S0, Sb, X:S' using E) >
   sol(replace(Cx:S0, P · (X:S' <- T:S')) · Sb)  .

rl < none ; mrew(P:S, Cx:S0, Sb, VSL) > => none .
```

When the subterm search tasks are exhausted, the whole `amatchrew` execution is discarded. Identical rules are used for the other variants, `matchrew` and `xmatchrew`, except for the initial one, where `genAmatch` should be replaced by the appropriate function.

Pruning of solutions. The semantics of the `one` combinator can be expressed using a trivial continuation `onec`:

```
rl < one(E) @ T:S > => < < E @ T:S > ; onec > .
rl < sol(T:S) TS ; onec > => sol(T:S) .
rl < none ; onec > => none .
```

Which solution is selected depends on the internal strategy of the rewriting engine for applying rules and ordering matches. Using the `search` command, every possible solution will be selected in some rewriting branch. The better performance is obtained if the second rule above is run just after the first solution appears inside the task, so that no unnecessary work is done. This is a situation where strategies are valuable.

Strategy modules and calls. Strategy modules, their declarations and definitions, can be represented as Maude terms, as already done for the metalevel in [1, §17.3]. For simplifying this presentation, we assume that the strategy definitions are collected in a definition set `DEFS`.

```
eq DEFS = (slabel(p1, ..., pn), δ, C) , ... .

rl < SC:StratCall @ T:S > => find-defs(DEFS, SC:StratCall, T:S) .

eq find-defs(none, SC, T:S) = none .
ceq find-defs((Slhs, Def, C) Defs, SC, T:S) =
    find-defs2(MAT, T:S, Def) find-defs(Defs, SC, T:S)
 if MAT := getMatch(Slhs, SC, C) .

eq find-defs2(none, T:S, Def) = none .
eq find-defs2(< Sb, Cx:S > MAT, T:S, Def) =
   < Def · Sb @ T:S > find-defs2(MAT, T:S, Def) .
```

The function `find-defs` traverses all the strategy definitions in `DEFS` and tries to match the strategy call term with their lefthand sides, and check their equational conditions. The strategy `find-defs2` takes these matches and produces a task `< Def · Sb @ T:S >` for each of them, to continue rewriting `T` with the definition strategy, whose free variables are bound according to the matching substitution.

5.1 Relation with the other semantics

This semantics is equivalent to the denotational semantics in the sense specified in the following proposition, i.e. they produce the same solutions and terminate for the same input data.

▸ **Theorem 1.** In any module (M, SM), for any term $t \in T_{\Sigma/E}$, and for any strategy expression α, $t' \in [\![\alpha]\!]_\Delta(\theta, t)$ iff $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>} \rightarrow^*_{\mathcal{S}(M,SM)} \texttt{sol(}t'\texttt{)}\ TS$ for some TS of sort `Tasks`. Moreover, $\bot \in [\![\alpha]\!]_\Delta(\theta, t)$ iff there is an infinite derivation from $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$ in $\mathcal{S}(M, SM)$.

Moreover, since the denotational semantics is in turn equivalent to the small-step operational semantics, the following corollary holds.

▸ **Corollary 2.** In any module (M, SM), for any term $t \in T_{\Sigma/E}$, and for any strategy expression α, $t \,@\, \alpha \rightarrow^*_{s,c} t' \,@\, \varepsilon$ iff $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>} \rightarrow^*_{\mathcal{S}(M,SM)} \texttt{sol(}t'\texttt{)}\ TS$ for some TS of sort `Tasks`. Moreover, there is an infinite derivation from $t \,@\, \alpha$ by $\rightarrow_{s,c}$ iff there is an infinite derivation from $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$ in $\mathcal{S}(M, SM)$.

A Proofs

A.1 Proofs for the semantic infrastructure

In this section, we prove that the infrastructure used to define the semantics is well defined. This covers the definitions and properties in Section 2.

▸ **Proposition 1.** For any set M, $\mathcal{P}_\perp(M)$ and $\leq$ are well defined. Classes are finite subsets of $M \cup \{\perp\}$ or pairs of infinite subsets $\{\perp\} \cup A$ and $A \backslash \{\perp\}$. The expressions $A \cup B$ and $x \in A$ are well defined for $x \in M$.

Remember $A \sim B \iff A = B \vee (A \oplus B = \{\perp\}$ and A is not finite) where $\oplus$ is the symmetric difference. In fact this is the same as $A \sim B \iff A = B \vee (A \oplus B = \{\perp\}$ and A and B are not finite) because A and B differ in a single element in this case.

We have to check $\sim$ is an equivalence relation. Reflexivity and symmetry are too simple. Let's prove reflexivity. Suppose $A \sim B$ and $B \sim C$. If $A = B$ or $B = C$ the proof is finished. Otherwise, the three sets are not finite and $A \oplus B = \{\perp\}$ and $B \oplus C = \{\perp\}$. Then $A \oplus C = \{\perp\}$.

To characterize classes, if A is finite then $A \sim B$ iff $A = B$ as the second condition does not hold. If A is infinite, it shares class with any infinite set B such that $A \oplus B = \{\perp\}$. If $\perp \in A$ the only such B is $A \backslash \{\perp\}$ and if $\perp \notin A$ its only companion is $\{\perp\} \cup A$.

$A \cup B$ is independent of the representatives for A and B, because finite sets has a unique representative and union with an infinite set produce an infinite set. $x \in A$ makes sense because both representatives contain the same elements except $\perp$.

The order is well defined too. This is clear if $\perp \notin A$. Otherwise, $A \backslash \{\perp\} \subseteq B$ makes sense because $A \backslash \{\perp\}$ is the same set, no matter which representative of A we take, and all representatives or none of B are contained in $A \backslash \{\perp\}$ because $\perp \notin A \backslash \{\perp\}$.

■

▸ **Proposition 2.** For any set M, $\mathcal{P}_\perp(M)$ is a chain-complete partial ordered set. Its minimum is $\{\perp\}$ and for any non-empty chain $F \subseteq \mathcal{P}_\perp(M)$

$$\sup F = \begin{cases} \bigcup_{A \in F} A & \text{if } \forall A \in F \;\; \perp \in A \\ Z & \text{if } \exists\, Z \in F \;\; \perp \notin Z \end{cases}$$

In the later case that Z such that $\perp \notin Z$ is unique.

We have to check that $\leq$ is a partial order and that every chain has a supremum in $\mathcal{P}_\perp(M)$. For the first task, let $A, B, C \in \mathcal{P}_\perp(M)$.

- *Reflexive*: we must prove $A \leq A$. If $\perp \in A$ then $A \leq A \iff A \backslash \{\perp\} \subseteq A$ which is true. Otherwise, $\perp \notin A$ so $A \leq A \iff A = A$.
- *Transitive*: suppose $A \leq B$ and $B \leq C$. If $\perp \notin A$ then $A = B$ and $\perp \notin B$. If $\perp \notin B$ then $B = C$ so $A \leq C$. Otherwise, $A \backslash \{\perp\} \subseteq B$ and $B \backslash \{\perp\} \subseteq C$, so $A \backslash \{\perp\} \subseteq C$, which means $A \leq C$.
- *Antisymmetric*: suppose $A \leq B$ and $B \leq A$. If $\perp \notin A$ or $\perp \notin B$ then $A = B$ by definition of $\leq$. Otherwise and also from the definition, $A \backslash \{\perp\} \subseteq B$ and $B \backslash \{\perp\} \subseteq A$. Both A and B contain $\perp$ so $A \subseteq B$ and $B \subseteq A$, and then $A = B$.

We can conclude $\leq$ is a partial order. Let F be a chain. We consider two cases:

- *For all $A \in F$, $\perp \in A$.* We prove $S = \bigcup_{A \in F} A \in \mathcal{P}_\perp(M)$ is the supremum.
  - It is an upper bound. Take $A \in F$ and see $A \leq S$. In this case $\perp \in A$, so this is equivalent to $A \backslash \{\perp\} \subseteq S$ and this is true, because $A \subseteq S$.
  - It is the least upper bound, because any other $B \in \mathcal{P}_\perp(M)$ such that $\forall A \in F \;\; A \leq B$ satisfies $S \leq B$ or equivalently $S \backslash \{\perp\} \subseteq B$ because $\perp \in S$. For any $x \in S \backslash \{\perp\}$, since S is a union, there is an $A \in F$ such that $x \in A$. Since $A \backslash \{\perp\} \subseteq B$, $x \in B$. We conclude $S \leq B$.

- *There is a $Z \in F$ such that $\perp \notin Z$*, then Z must be the supremum of F. For this to make sense there must be only one set such that $\perp \notin Z$. Suppose there were another set $\perp \notin A \in F$. Then $A \leq Z$ or $Z \leq A$ because F is a chain, in any case $A = Z$. There is only one.
  - Z is an upper bound. Take $A \in F$, let's see $A \leq Z$. F is a chain so $A \leq Z$ or $Z \leq A$. If the first holds we have finished. Otherwise, $Z = A$ provided $\perp \notin Z$ and consequently $A \leq Z$ too.
  - Since $Z \in F$, it must be the supremum.

■

▸ **Lemma 3.** For any set M and $A, B, C \in \mathcal{P}_\perp(M)$ and a chain $F \subseteq P_\perp(M)$

1. If $B \leq C$, $A \cup B \leq A \cup C$.
2. If $\perp \in A$ and $B \subseteq C$, $A \cup B \leq A \cup C$.
3. $\sup\{A \cup B : A \in F\} = (\sup F) \cup B$
4. $\bigcup_{A \in F} A \leq \sup F \subseteq \bigcup_{A \in F} A$.
5. $\bigcup_{A \in F} A \backslash \{\perp\} \subseteq \sup F \leq \bigcup_{A \in F} A \backslash \{\perp\}$.
6. $\sup F = \bigcup_{A \in F} A \backslash \{\perp\}$ if $\exists A \in F \quad \perp \notin A$.

▸ **Definition.** For any sets M, N, we define on $M \to \mathcal{P}_\perp(N)$ the order $f \sqsubseteq g \iff \forall x \in M \;\; f(x) \leq g(x)$.

▸ **Proposition 4.** $(M \to \mathcal{P}_\perp(N), \sqsubseteq)$ is a ccpo and for any family $F \subseteq M \to \mathcal{P}_\perp(N)$

$$(\sup F)(x) = \sup F(x) = \sup\{f(x) : f \in F\}$$

Hence, for any non-empty set $S \subseteq M$, $\sup\{f|_S : f \in F\} = (\sup F)|_S$.

▸ **Definition.** For any sets M and N

1. We say $A \subseteq \mathcal{P}_\perp(N)$ is *final* if $\perp \notin A$.
2. We say $f : M \to \mathcal{P}_\perp(N)$ is *final* if $f(x)$ is final for all $x \in M$.
3. Any chain F in $P_\perp(N)$ is *final* if it contains a final element.
4. Any chain F in $M \to P_\perp(N)$ is *final* if $F(x)$ is final for all $x \in M$.

A final chain in $M \to P_\perp(N)$ need not contain a final element, i.e. a function whose images are all final. Consider $M = \mathbb{N}$, $N = \emptyset$, and $F = \{f_n : n \in \mathbb{N}\}$ with $f_n(k) = \{\perp\}$ for $k \geq n$ and $f_n(k) = \emptyset$ for $k < n$. It is a chain because $f_n \sqsubseteq f_{n+1}$ and it is final since $F(k)$ contains $\emptyset$. However, none of the f_n is final.

▸ **Proposition 5.** Given sets M, N, a finite subset $S \subseteq M$, and a non-empty chain $F \subseteq M \to \mathcal{P}_\perp(N)$,

$$\forall x \in S \;\; F(x) \text{ is final} \quad \Longrightarrow \quad \exists f \in F \;\; \forall x \in S \;\; f(x) \text{ is final} \quad (= \sup F(x))$$

We prove the statement by induction on the size of S.

- If $S = \emptyset$, then the statements reads $\exists f \in F$, which is true.
- If $S = \{x\}$, then $F(x)$ is final and there is a final set $f(x) \in F(x)$. We have found f.
- Otherwise, we can split S in two non-empty parts A and B such that $A \cup B = S$. By the induction hypothesis, there is an $f \in F$ such that $\perp \notin f(x)$ for all $x \in A$ and $g \in F$ such that $\perp \notin g(x)$ for all $x \in B$. Since F is a chain, $f \sqsubseteq g$ or $g \sqsubseteq f$. Without loss of generality, assume $f \sqsubseteq g$ and

  $$f \sqsubseteq g \;\Rightarrow\; \forall x \in S \;\; f(x) \leq g(x) \;\Rightarrow\; \forall x \in A \;\; f(x) = g(x) \;\Rightarrow\; \forall x \in A \;\; g(x) \text{ is final}$$

  Since $g(x)$ is already final for all $x \in B$, g is final.

■

▸ **Proposition 6.** For any indexed families $\{A_i\}_{i\in I}, \{B_i\}_{i\in I} \subseteq \mathcal{P}_\perp(M)$ such that $A_i \leq B_i$,

$$\bigcup_{i\in I} A_i \leq \bigcup_{i\in I} B_i$$

We split the union depending on whether $\perp \in A_i$ or not. If $\perp \in A_i$, $A_i \leq B_i$ means $A_i\backslash\{\perp\} \subseteq B_i$. Then

$$\left(\bigcup_{i\in I,\perp\in A_i} A_i\right)\backslash\{\perp\} = \bigcup_{i\in I,\perp\in A_i} A_i\backslash\{\perp\} \subseteq \bigcup_{i\in I,\perp\in A_i} B_i \qquad \text{so} \qquad \bigcup_{i\in I,\perp\in A_i} A_i \leq \bigcup_{i\in I,\perp\in A_i} B_i$$

On the other hand, if $\perp \notin A_i$ then $A_i = B_i$. All together,

$$\bigcup_{i\in I} A_i = \bigcup_{i\in I,\perp\notin A_i} A_i \cup \bigcup_{i\in I,\perp\in A_i} A_i = \bigcup_{i\in I,\perp\notin A_i} B_i \cup \bigcup_{i\in I,\perp\in A_i} A_i \overset{\text{(L3:1)}}{\leq} \bigcup_{i\in I,\perp\notin A_i} B_i \cup \bigcup_{i\in I,\perp\in A_i} B_i = \bigcup_{i\in I} B_i$$

■

▸ **Proposition 7.** For any sets M, N and finite $S \subseteq M$, and a chain $F \subseteq M \to \mathcal{P}_\perp(N)$

$$\sup\left\{\bigcup_{x\in S} f(x) : f \in F\right\} = \bigcup_{x\in S} (\sup F)(x)$$

And if S is infinite and the previous does not hold, $\sup\left\{\bigcup_{x\in S} f(x) : f \in F\right\}\backslash\{\perp\} = \bigcup_{x\in S}(\sup F)(x)$.

Let L be the set to which the sup is applied in the left-hand side. For the proposition to make sense, L must be a chain. But given $f, g \in F$ we have $f \sqsubseteq g$ or $g \sqsubseteq f$. Without loss of generality, suppose the first holds. Then $f(x) \leq g(x)$ for all $x \in M$. According to Proposition 6, $\bigcup_{x\in S} f(x) \leq \bigcup_{x\in S} g(x)$. L is a chain.

To check the equality, we proceed differently whether F is final or not. If it is not, there is a $y \in S$ such that $F(y)$ is not final. Hence, $\perp \in \bigcup_{x\in M} f(x) \supseteq f(y)$ for all $f \in F$ and then the supremum of L is

$$\sup L = \bigcup_{f\in F}\bigcup_{x\in S} f(x) = \bigcup_{x\in S}\bigcup_{f\in F} f(x)$$

according to Proposition 2. From the other side $(\sup F)(y) = \bigcup_{f\in F} f(y)$ so $\perp \in (\sup F)(y)$ and as consequence $\perp$ is in the right-hand side union. Then

$$\bigcup_{x\in S}(\sup F)(x) = \{\perp\} \cup \bigcup_{x\in S}(\sup F)(x)\backslash\{\perp\} \overset{\text{(L3:6)}}{=} \{\perp\} \cup \bigcup_{x\in S}\bigcup_{f\in F} f(x)\backslash\{\perp\} = \bigcup_{x\in S}\bigcup_{f\in F} f(x)$$

Then both sides coincide.

If $F(x)$ is final for all $x \in S$ and S is finite, by Proposition 5, there is a $Z \in F$ such that $Z(x)$ is final for $x \in S$. Therefore, $Z(x) = (\sup F)(x)$ for all $x \in S$, and then $\bigcup_{x\in S}(\sup F)(x)$ is in L and is a final set because $\perp$ does not belong to any set in the union, so it is the supremum of L.

Even if S is infinite, most of the previous proof is still valid. The finiteness of S was only used to claim the existence of a final $Z \in F$. Thus, the only case which has not been covered is when F is final but there is not a final function in F. Then, every element in L contains $\perp$ and the supremum can be calculated as in the non-final case. Then

$$\bigcup_{x\in S}(\sup F)(x) \overset{\text{(L3:6)}}{=} \bigcup_{x\in S}\bigcup_{f\in F} f(x)\backslash\{\perp\} = \left(\bigcup_{x\in S}\bigcup_{f\in F} f(x)\right)\backslash\{\perp\} = \sup L\backslash\{\perp\}$$

■

We use the auxiliary function $bot(A) = \{\perp\}$ **if** $\perp \in A$ **else** $\emptyset$, to avoid some case distinctions in the following.

▸ **Lemma 8.** For any sets M and N, and $A, B \in \mathcal{P}_\perp(M)$ and $C_x \in \mathcal{P}_\perp(N)$ for all $x \in M$

1. If $A \subseteq B$ then $\mathsf{let}\ x \leftarrowtail A : C_x \subseteq \mathsf{let}\ x \leftarrowtail B : C_x$.
2. $(\mathsf{let}\ x \leftarrowtail A : B_x) \backslash \{\perp\} \subseteq \bigcup_{x \in A\backslash\{\perp\}} B_x$.
3. $\mathsf{let}\ x \leftarrowtail A : B_x = bot(A) \cup \mathsf{let}\ x \leftarrowtail A\backslash\{\perp\} : B_x$
4. $\{\perp\} \cup \bigcup_{x \in A\backslash\{\perp\}} B_x \leq \mathsf{let}\ x \leftarrowtail A : B_x \leq \bigcup_{x \in A\backslash\{\perp\}} B_x$
5. $\bigcup_{y \in I} \mathsf{let}\ x \leftarrowtail A : C_{x,y} = \mathsf{let}\ x \leftarrowtail A : \bigcup_{y \in I} C_{x,y}$.

▸ **Lemma 9.** For any sets M and N, and $A, B \in \mathcal{P}_\perp(M)$ and $C_x, D_x \in \mathcal{P}_\perp(N)$ for all $x \in M$.

1. If $A \leq B$ then $\mathsf{let}\ x \leftarrowtail A : C_x \leq \mathsf{let}\ x \leftarrowtail B : C_x$.
2. If $C_x \leq D_x$ for all $x \in M$ then $\mathsf{let}\ x \leftarrowtail A : C_x \leq \mathsf{let}\ x \leftarrowtail A : D_x$.
   The premise is equivalent to $C \sqsubseteq D$, being $C, D : M \to \mathcal{P}_\perp(N)$.

1. If A is final, $A = B$ and the inequality follows from reflexivity. Otherwise, $\perp \in A$ implies $A\backslash\{\perp\} \subseteq B$ and in particular $A\backslash\{\perp\} \subseteq B\backslash\{\perp\}$. From set theory $\bigcup_{x \in A\backslash\{\perp\}} C_x \subseteq \bigcup_{x \in B\backslash\{\perp\}} C_x$ and then

$$\perp \in \mathsf{let}\ x \leftarrowtail A : C_x = \{\perp\} \cup \bigcup_{x \in A\backslash\{\perp\}} C_x \overset{\text{(L3:1)}}{\leq} \{\perp\} \cup \bigcup_{x \in B\backslash\{\perp\}} C_x \overset{\text{(L8:4)}}{\leq} \mathsf{let}\ x \leftarrowtail B : C_x$$

2. We have

$$\mathsf{let}\ x \leftarrowtail A : C_x = bot(A) \cup \bigcup_{x \in A} C_x \overset{\text{(P6)}}{\leq} bot(A) \cup \bigcup_{x \in A} D_x = \mathsf{let}\ x \leftarrowtail A : D_x$$

■

▸ **Lemma 10.** For any sets M and N, and $A \in \mathcal{P}_\perp(M)$ and $C_x \in \mathcal{P}_\perp(N)$ for all $x \in M$.

1. For any chain $F \subseteq \mathcal{P}_\perp(M)$, $\mathsf{let}\ x \leftarrowtail \sup F : C_x = \sup \{\, \mathsf{let}\ x \leftarrowtail A : C_x \ :\ A \in F\}$.
2. For any chain $F \subseteq (M \to \mathcal{P}_\perp(N))$, $\mathsf{let}\ x \leftarrowtail A : \sup F = \sup \{\, \mathsf{let}\ x \leftarrowtail A : C_x \ :\ C \in F\}$.

The right-hand side of the equation of each statement is well defined because let, being monotone, transforms chains into chains.

1. First, suppose F is final, then there is a $Z \in F$ such that $\sup F = Z \not\ni \perp$. We must prove $S := \mathsf{let}\ x \leftarrowtail Z : C_x$ is the supremum of the right-hand side chain.

   Because $Z \in F$, S is in the chain and proving that it is an upper bound is enough. But this come from monotonicity as for any $B \in F$ we have $B \leq Z$ and then $\mathsf{let}\ x \leftarrowtail B : C_x \leq \mathsf{let}\ x \leftarrowtail Z : C_x = S$ by the previous lemma.

   Now, suppose F is not final, $\forall A \in F\ \ \perp \in A$. Then

$$\mathsf{let}\ x \leftarrowtail \bigcup_{A \in F} A : B_x \overset{\text{(P2)}}{=} \{\perp\} \cup \bigcup_{x \in \left(\bigcup_{A \in F} A\right)\backslash\{\perp\}} B_x = \{\perp\} \cup \bigcup_{A \in F} \bigcup_{x \in A\backslash\{\perp\}} B_x = \bigcup_{A \in F} \left(\{\perp\} \cup \bigcup_{x \in A\backslash\{\perp\}} B_x \right)$$
$$= \bigcup_{A \in F} \mathsf{let}\ x \leftarrowtail A : B_x = \sup \{\, \mathsf{let}\ x \leftarrowtail A : C_x \ :\ C \in F\}$$

2. Our goal is to prove the following (we have used Lemma 8 between lines)

$$\begin{array}{ccc} \mathsf{let}\ x \leftarrowtail A : \sup F & = & \sup\{\,\mathsf{let}\ x \leftarrowtail A : C_x \ :\ C \in F\} \\ = & & =^{(\text{L3:3})} \\ bot(A) \cup \bigcup_{x\in A\setminus\{\bot\}} (\sup F)_x & & bot(A) \cup \sup\left\{\bigcup_{x\in A\setminus\{\bot\}} C_x : C \in F\right\} \end{array}$$

If $\bot \notin A$ then $bot(A) = \emptyset$ and A is finite. The straightforward application of Proposition 7 give us the result. Otherwise $\bot \in A$ and $bot(A) = \{\bot\}$. We can still apply the same proposition and, being A finite or not, we obtain the equality because $\{\bot\}$ is in both sides.

■

Up to now we have proven some properties of $\mathcal{P}_\bot(M)$ and its let operator. This section ends by stating some standard facts about semantic functions defined as in Section 2.1, and set the stage for the proofs for Section 3 below. The following proposition proof is omitted.

▸ **Proposition 11.** The following sets are ccpo

1. SFun with order $f \sqsubseteq g \iff \forall(\theta,t) \in \mathrm{SDom} \quad f(\theta,t) \le g(\theta,t)$.
   And for any chain $F \subseteq$ SFun, its supremum is $(\sup F)(\theta,t) = \sup\{f(\theta,t) : f \in F\}$.
2. SFun^n with order $(f_1,\dots,f_n) \sqsubseteq (g_1,\dots,g_n) \iff \forall(\theta,t) \in \mathrm{SDom} \quad \bigwedge_{i=1}^n f_i(\theta,t) \le g_i(\theta,t)$.
   And for any chain $F \subseteq \mathrm{SFun}^n$, its supremum is $\sup F = (\sup\{f_i : (f_1,\dots,f_n) \in F\})_{i=1}^n$.
3. $\mathrm{SFun}^n \to$ SFun with order $F \sqsubseteq G \iff \forall f_1,\dots,f_n \in \mathrm{SFun} \quad F(f_1,\dots,f_n) \sqsubseteq G(f_1,\dots,f_n)$.
   And any chain C has supremum $(\sup C)(f_1,\dots,f_n)(\theta,t) = \sup\{F(f_1,\dots,f_n)(\theta,t) : F \in C\}$.
4. $\mathrm{SFun}^n \to \mathrm{SFun}^m$ with order $F \sqsubseteq G \iff \bigwedge_{i=1}^m F_i \sqsubseteq G_i$ where F_i is the i-th component of F.

▸ **Lemma 12.** A function $\mathrm{SFun}^n \to$ SFun is monotonic if and only if it is monotonic on every coordinate. It is continuous if and only if it is continuous on every coordinate.

A.2 Proofs for the denotational semantics

Hereinafter, we prove that the denotational semantics in Section 3 is well defined. The key point is that the fixed point used to define d_i exists. Kleene Fixed Point Theorem will be used for that purpose: $\mathcal{P}_\bot(M)$ must be a chain-complete partial ordered set (as we have proved in the previous section) and the semantic functions must be monotonic and (Scott) continuous. In the following, we omit the definition context Δ subscript in the semantic function, since it is understood from the context.

▸ **Lemma 13.** The following functions $\mathrm{SFun}^n \to$ SFun are continuous for any rule label rl, $l, r, P \in T_\Sigma(X)$ and condition C:

1. $\mathit{Comp}(f,g) = g \circ f$
2. $\mathit{Union}(f,g) = (\theta,t) \mapsto f(\theta,t) \cup g(\theta,t)$
3. $\mathit{Star}(f) = (\theta,t) \mapsto \bigcup_{n=0}^\infty f^n(\theta,t) \cup \{\bot : f^n(\theta,t) \neq \emptyset \text{ for all } n \in \mathbb{N}\}$
4. $\mathit{Cond}(f,g,h) = (\theta,t) \mapsto h(\theta,t)$ **if** $f(\theta,t) = \emptyset$ **else** $g \circ f(\theta,t)$
5. $\mathit{Check}(C,\theta,f_1,\dots,f_n) = \mathsf{check}(C,\theta,\alpha_1 \cdots \alpha_n)$ with $[\![\alpha_i]\!]$ replaced by f_i. And its derived operators *Mcheck*, *Xcheck* and *Amcheck*.
6. $\mathit{ApplyRule}(f_1,\dots,f_n) = (\theta,t) \mapsto \bigcup_{(rl,l,r,C)\in R}$ $\mathsf{let}\ (\sigma,c) \leftarrowtail \mathit{Amcheck}(l,t,$
   $C, \mathrm{id}[x_1/\theta(t_1),\dots,x_n/\theta(t_n)]; f_1 \cdots f_m, \theta_s) : \{c(\sigma(r))\}$.

7. $\mathit{SubRewrite}(f_1, \dots, f_n) = (\theta, t) \mapsto \bigcup_{\sigma \in \mathsf{mcheck}(P,t,C,\theta)}$
$$\mathsf{let}\ t_1 \leftarrowtail f_1(\sigma, \sigma(x_1)), \dots, t_n \leftarrowtail f_n(\sigma, \sigma(x_n)) : \{\sigma[x_1/t_1, \dots, x_n/t_n](P)\}.$$

According to Lemma 12, it is enough to prove that $F : \mathrm{SFun}^n \to \mathrm{SFun}$ is monotonic and continuous by coordinates. Remember that $F : \mathrm{SFun} \to \mathrm{SFun}$ is monotonic if $F(f)(\theta, t) \leq F(g)(\theta, t)$ whenever $f \sqsubseteq g$ and F is continuous if $\sup\{F(f)(\theta, t) : f \in C\} = F(\sup C)(\theta, t)$ for all $(\theta, t) \in \mathrm{SDom}$.

1. $\mathit{Comp}(f, g)(\theta, t) = \mathsf{let}\ t' \leftarrowtail f(\theta, t) : g(\theta, t')$. This is already solved using Lemma 9 for the monotonicity and Lemma 10 for continuity.

2. $\mathit{Union}(f, g)(\theta, t) = f(\theta, t) \cup g(\theta, t)$. This finite union is monotonic by Proposition 6 and continuous by Proposition 7.

3. $\mathit{Star}(f)(\theta, t) = \bigcup_{n \in \mathbb{N}} f^n(\theta, t) \cup \{\bot : f^n(\theta, t) \neq \emptyset \text{ for all } n \in \mathbb{N}\}$. First, f^n is monotonic and continuous after a simple induction on n provided $\circ$ or *Comp* are. The monotonicity and continuity of *Star* follows from those of the two clauses separately, by the same arguments given for the *Union* case. Let S be the second clause as a function of f for some fixed θ and t, i.e. $S(f) = \{\bot\}$ if $f^n(\theta, t) \neq \emptyset$ for all $n \in \mathbb{N}$, $S(f) = \emptyset$ otherwise. S is monotonic, since $S(f) = \emptyset$ implies there is an $n \in \mathbb{N}$ such that $f^n(\theta, t) = \emptyset$, so $g^n(\theta, t) = \emptyset$ for all $g \sqsupseteq f$ by the monotonicity of exponentiation, and so $S(g) = \emptyset$. Moreover, for any chain F, if $S(\sup F) = \emptyset$, there must be an $n \in \mathbb{N}$ such that $(\sup F)^n(\theta, t) = \emptyset$. Thus, by the continuity of exponentiation, there is an $f \in F$ that $f^n(\theta, t) = \emptyset$ and then $S(f) = \emptyset$. This implies $\sup S(F) = \emptyset$ and so S is continuous.

   For the first clause, monotonicity comes from Proposition 6. Proving continuity is not as straightforward, but follows from Proposition 7. Given a chain $F \subseteq \mathrm{SFun}$ and a pair (θ, t), we invoke this proposition with $S = M = \mathbb{N}$ and the chain $F' = \{n \mapsto f^n(\theta, t) : f \in F\}$. In fact, it is a chain because if $f \sqsubseteq g$ then $f^n \sqsubseteq g^n$ for all $n \in \mathbb{N}$. Its supremum is the function $\sup F' = n \mapsto (\sup F)^n(\theta, t)$. Since $\mathbb{N}$ is infinite the proposition says

$$\sup\left\{\bigcup_{n=0}^{\infty} f^n(\theta, t)\right\} = \bigcup_{n=0}^{\infty} (\sup C)^n(\theta, t) \qquad \vee \qquad \sup\left\{\bigcup_{n=0}^{\infty} f^n(\theta, t)\right\} \setminus \{\bot\} = \bigcup_{n=0}^{\infty} (\sup C)^n(\theta, t)$$

   If the first holds, then *Star* is continuous. If the sets are infinite, even the second means that *Star* is continuous. It remains to see that the first must hold if $R := \bigcup_{n=0}^{\infty} (\sup C)^n(\theta, t)$ is finite.

   If $\bot \in R$ then the second cannot hold, so the first does. Otherwise, $\bot \notin R$ and the second may be true. In this case F must be final in R. Suppose it is not, i.e. that exists a $t' \in R$ such that $\bot \in (\sup F)(\theta, t')$. Since $t' \in R$, $t' \in (\sup F)^n(\theta, t)$ for some $n \geq 1$ and then $\bot \in (\sup F)^{n+1}(\theta, t) \subseteq R$. And this is a contradiction.

   According to Proposition 5, there is a function $z \in C$ such that $z(\theta, u) = (\sup C)(\theta, u)$ for all $u \in R$, which is finite. We will prove that the element $\bigcup_{n=0}^{\infty} z^n(\theta, t)$ of the left-hand side chain is contained in R. In that case, since $\bot \notin R$, this set is final and then $\bot$ is not in the chain's supremum, so the first identity holds and *Star* is continuous.

   The proof is done by induction on n. The base case is $n = 0$, where $z^0(\theta, t) = \{t\} \subseteq R$. Suppose $z^n(\theta, t) \subseteq R$, we know $z^{n+1}(\theta, t) = \mathsf{let}\ t' \leftarrowtail z^n(\theta, t) : z(\theta, t')$. Since $t' \in R$, there is an $m \geq 1$ such that $t' \in (\sup C)^m(\theta, t)$. Then $z(\theta, t') = (\sup C)(\theta, t') \subseteq (\sup C)^{m+1}(\theta, t) \subseteq R$. The union for all $t \in z^n(\theta, t)$ is $z^{n+1}(\theta, t)$ and it is in R (no $\{\bot\}$ is added by let because $\bot \notin z^n(\theta, t) \subseteq R$).

4. Monotony and continuity of *Cond* in g and h with f fixed is clear, because the function is then a composition or the identity in h. We should only take care of f changing with fixed g and h.

   For $f, f' \in \mathrm{SFun}$ and $f \sqsubseteq f'$. If $f(\theta, t) = \emptyset$ then $f'(\theta, t) = \emptyset$ and nothing changes. If $f'(\theta, t) = \emptyset$, $f(\theta, t)$ can only be $\emptyset$ (just seen) or $\{\bot\}$, but in this case

$$\mathit{Cond}(f, g, h)(\theta, t) = (g \circ f)(\theta, t) = \mathsf{let}\ t' \leftarrowtail \{\bot\} : g(\theta, t') = \{\bot\}$$

   so it is less or equal than $\mathit{Cond}(f', g, h)$ no matter what it is. In any other case $\mathit{Cond}(f, g, h)(\theta, t) = \mathit{Comp}(f, g)(\theta, t)$ which is monotone and continuous in f.

5. We check monotonicity and continuity by induction on the structure of C.

- The base case is $\mathit{Check}(\texttt{true}, \theta, f_1, \dots, f_n, \theta_s) = \{\theta\}$ which is a constant in f_1 to f_n. Hence, it is monotonic and continuous.
- For $\mathit{Check}(l = r \wedge C, \theta, f_1, \dots, f_n, \theta_s)$. If $\theta(l) = \theta(r)$ that is $\mathit{Check}(C, \theta, f_1, \dots, f_n, \theta_s)$. By induction hypothesis it is continuous. In the other case, their value is $\emptyset$, a constant, which is continuous.
- For $\mathit{Check}(l := r \wedge C, \theta, f_1, \dots, f_n, \theta_s) = \bigcup_{\sigma \in \mathsf{match}(\theta(l), \theta(r))} \mathit{Check}(C, \sigma \circ \theta, f_1, \dots, f_n, \theta_s)$. By induction hypothesis all recursive calls to check are monotone and continuous. The finite union of continuous functions is continuous.
- For the rewriting condition,

$$\mathit{Check}(l \Rightarrow r \wedge C, \theta, f_1, \dots, f_n, \theta_s) = \\ \mathsf{let}\ t \leftarrowtail f_1(\theta, r) : \bigcup_{\sigma \in \times\mathsf{match}(\theta(r), t)} \mathit{Check}(C, \sigma \circ \theta, f_2 \cdots f_n, \theta_s)$$

  By induction hypothesis, the union terms are continuous. Their finite union is continuous, and the let is also continuous in f_1 in the first argument and $f_2, \dots, f_n$ in the second one.

The derived check operators are finite unions of all substitutions from a match so, by Proposition 7, they are continuous.

6. For the rule application, $\mathit{Amcheck}(l, t, C, \mathrm{id}[x_1/\theta(t_1), \dots, x_n/\theta(t_n)], f_1 \cdots f_n, \theta_s)$ is continuous in all f_i. Then by the arguments used above the let is continuous for any substitution, and the union of those is also continuous.
7. The f_i are arguments in the *index* part of the let construct, then the let are continuous in $f_1, \dots, f_n$. The finite union of all those is continuous in $f_1, \dots, f_n$.

■

▸ **Theorem 14.** Given a rewrite theory $\mathcal{R} = (\Sigma, E \uplus Ax, R)$ with strategies (G, D), the functional $F : \mathrm{SFun}^m \to \mathrm{SFun}^m$ given by

$$F(d_1, \dots, d_m) := (\llbracket \delta_1 \rrbracket(d_1, \dots, d_m), \dots, \llbracket \delta_m \rrbracket(d_1, \dots, d_m))$$

is well defined, monotonic and continuous. Then F has a least fixed point

$$\mathsf{FIX}\, F = \sup \{F^n(\{\bot\}, \dots, \{\bot\}) : n \in \mathbb{N}\}$$

All semantic functions are well defined because they are given an unambiguous functional definition and all cases are covered.

According to Kleene's Fixed Point Theorem [13, Thm 5.11] [3, Thm 4.3.6], if F is continuous in SFun^m then it has a least fixed point calculated as above. In fact, monotonicity is a sufficient condition for the existence of a least fixed point according to Knaster-Tarski Fixed Point Theorem [13, Sec 5.5] [3, Thm 2.3.2], but not to find the characterization we have written.

F is monotonic and continuous if each component $F_i = \llbracket \delta_i \rrbracket$ is (see Proposition 11:4). We will prove it by induction on δ_i structure:

- For `idle`, `fail`, and the `match` alternatives, the semantic function is a constant on d_i. Then it is continuous.
- $\llbracket \alpha\ ;\ \beta \rrbracket = \mathit{Comp}(\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket)$. By induction hypothesis $\llbracket \alpha \rrbracket$ and $\llbracket \beta \rrbracket$ are monotonic and continuous. *Comp* is continuous by the previous lemma, and functional composition preserves both. Hence, *Comp* is continuous.
- Provided $\llbracket \alpha \,|\, \beta \rrbracket = \mathit{Union}(\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket)$, $\llbracket \alpha^* \rrbracket = \mathit{Star}(\llbracket \alpha \rrbracket)$ and $\llbracket \alpha\ ?\ \beta\ :\ \gamma \rrbracket = \mathit{Cond}(\llbracket \alpha \rrbracket, \llbracket \beta \rrbracket, \llbracket \gamma \rrbracket)$, the same argument applies.

- Rule application and matchrew semantics are given by their corresponding functions *ApplyRule* and *SubRewrite*, which are continuous by the previous lemma. The first is $\mathit{ApplyRule}([\![\alpha_1]\!], \dots, [\![\alpha_n]\!])$ and the matchrew semantics is $\mathit{SubRewrite}([\![\alpha_1]\!], \dots, [\![\alpha_n]\!])$, and $[\![\alpha_i]\!]$ are continuous by induction hypothesis in $d_1, \dots, d_n$. Hence, both are continuous.
- For $[\![sl(t_1, \dots, t_n)]\!](\theta, t) = f_{sl}(\theta(t_1), \dots, \theta(t_n), t)$. And

$$f_{sl}(\vec{s}, t) = \bigcup_{(sl, \vec{p}_i, \delta_i, C_i) \in D} \bigcup_{\sigma \in \mathsf{vmatch}(\vec{p}_i, \vec{s}, C_i)} d_i(\sigma, t)$$

  is monotonic and continuous because it is the finite (and the indices do not depend on d_i) union of some d_i, and the identity is clearly monotonic and continuous.

■

▸ **Corollary 15.** Given a rewrite theory as in the previous theorem, the semantics of any strategy term is well defined and $d_i = [\![\delta_i]\!]$.

A.2.1 About the alternative semantics without scopes

▸ **Proposition 16.** Given a rewrite theory with strategies $\mathcal{R}$. For any strategy $\alpha \in \mathrm{Strat}_{\mathcal{R},\Omega}$, substitution $\theta : X \to T_\Sigma(X)$ and term $t \in T_\Sigma(X)$

$$[\![\alpha]\!]_\Delta(\theta, t) = [\![\theta(\alpha)]\!]_\Delta(\mathrm{id}, t)$$

First, we prove that

$$\mathsf{check}(\theta(C), \mathrm{id}; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \theta = \mathsf{check}(C, \theta; \alpha_1 \cdots \alpha_k, \theta_s)$$

assuming $[\![\alpha_i]\!](\theta, t) = [\![\theta(\alpha_i)]\!](\mathrm{id}, t)$ for $i \in \{1, \dots, k\}$ by induction on C, where $\Theta \circ \theta = \{\sigma \circ \theta : \sigma \in \Theta\}$.

- The base case is $C = \mathtt{true} = \theta(C)$. Then $\mathsf{check}(\mathtt{true}, \theta; \alpha_1 \cdots \alpha_k, \theta_s) = \{\theta\} = \{\mathrm{id}\} \circ \theta$.
- For $l = r \wedge C$, in both cases we obtain the condition $\theta(l) = \theta(r)$. When the condition does not hold, we obtain $\emptyset$ in both sides. Otherwise and by the hypothesis, both sides coincide.
- For $l : s \wedge C$, the proof is almost identical.
- For $l := r \wedge C$, we have

$$\begin{aligned}\mathsf{check}(\theta(l) &:= \theta(r) \wedge \theta(C), \mathrm{id}; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \theta \\ &= \bigcup_{\sigma \in \mathsf{match}(\theta(l), \theta(r))} \mathsf{check}(\theta(C), \sigma; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \theta\end{aligned}$$

  By the induction hypothesis, in one direction,

$$\mathsf{check}(\theta(C), \sigma; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) = \mathsf{check}(\sigma(\theta(C)), \mathrm{id}; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \sigma$$

  and in the other direction

$$\mathsf{check}((\sigma \circ \theta)(C), \mathrm{id}; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \sigma \circ \theta = \mathsf{check}(C, \sigma \circ \theta; \alpha_1 \cdots \alpha_k, \theta_s)$$

  Changing the union terms, we obtain the right-hand side of the desired identity.
- Finally, for $l \Rightarrow r \wedge C$.

$$\begin{aligned}&\mathsf{check}(\theta(l) \Rightarrow \theta(r) \wedge \theta(C), \mathrm{id}; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \theta \\ &\quad = \mathsf{let}\ t \leftarrowtail [\![\theta_s(\alpha_1)]\!](\mathrm{id}, \theta(l)) : \bigcup_{\sigma \in \mathsf{xmatch}(\theta(r), t)} \mathsf{check}(\theta(C), \sigma; \theta_s(\alpha_1) \cdots \theta_s(\alpha_k), \mathrm{id}) \circ \theta\end{aligned}$$

  From hypothesis, we know $[\![\theta_s(\alpha_1)]\!](\mathrm{id}, \theta(l)) = [\![\alpha_1]\!](\theta_s, \theta(l))$. The same pair of applications of the induction hypothesis in the previous case gives us the right-hand side term.

Then, we have

$$\begin{aligned}\mathsf{mcheck}(p,t,C,\theta;\alpha_1\cdots\alpha_n,\theta_s) &= \bigcup_{\sigma\in \texttt{match}\ P\ \texttt{s.t}\ C(\theta(p),t)} \mathsf{check}(C,\sigma\circ\theta,\alpha_1\cdots\alpha_n,\theta_s)\\ &= \bigcup_{\sigma\in \texttt{match}\ P\ \texttt{s.t}\ C(\theta(p),t)} \mathsf{check}(\theta(C),\sigma,\theta_s(\alpha_1)\cdots\theta_s(\alpha_n),\mathrm{id})\circ\theta\\ &= \mathsf{mcheck}(\theta(p),t,\theta(C),\mathrm{id};\theta_s(\alpha_1)\cdots\theta_s(\alpha_n),\mathrm{id})\circ\theta\end{aligned}$$

Notice that in the third equality we have rewritten the union terms twice, in different directions.

Now, we prove "forall θ and t, $[\![\alpha]\!](\theta,t) = [\![\theta(\alpha)]\!](\mathrm{id},t)$" by induction on the structure of α.

- For `idle`, `fail`, we have $\theta(\alpha)=\alpha$. The statement is trivially true .
- For $\texttt{match}\ P\ \texttt{s.t}\ C$, we have $[\![\texttt{match}\ P\ \texttt{s.t}\ C]\!](\theta,t) = \{t\}$ **if** $\mathsf{mcheck}(P,t,C,\theta)\neq\emptyset$ **else** $\emptyset$.
  From the other side $[\![\theta(\texttt{match}\ P\ \texttt{s.t}\ C)]\!](\mathrm{id},t) = [\![\texttt{match}\ \theta(P)\ \texttt{s.t}\ \theta(C)]\!](\mathrm{id},t) = \{t\}$ if the condition holds $\mathsf{mcheck}(P,t,\theta(C),\mathrm{id})\neq\emptyset$ or else $\emptyset$. Using what we have proved for mcheck, we have finished.
- For $sl(t_1,\ldots,t_n)$, then $\theta(sl(t_1,\ldots,t_n)) = sl(\theta(t_1),\ldots,\theta(t_n))$

  $$[\![\theta(sl(t_1,\ldots,t_n))]\!](\mathrm{id},t) = [\![sl(\theta(t_1),\ldots,\theta(t_n))]\!](\mathrm{id},t) = f_{sl}(\theta(t_1),\ldots,\theta(t_n),t) = [\![sl(t_1,\ldots,t_n)]\!](\theta,t)$$

- For $\alpha\ ;\ \beta$, then $\theta(\alpha\ ;\ \beta) = \theta(\alpha)\ ;\ \theta(\beta)$ and using induction hypothesis on α and β

  $$\begin{aligned}[\![\theta(\alpha\ ;\ \beta)]\!](\mathrm{id},t) = [\![\theta(\alpha)\ ;\ \theta(\beta)]\!](\mathrm{id},t) = [\![\theta(\beta)]\!]\circ[\![\theta(\alpha)]\!](\mathrm{id},t) = \mathsf{let}\ t' \leftarrowtail [\![\theta(\alpha)]\!](\mathrm{id},t) : [\![\theta(\beta)]\!](\mathrm{id},t')\\ \mathsf{let}\ t' \leftarrowtail [\![\alpha]\!](\theta,t) : [\![\beta]\!](\theta,t') = [\![\beta]\!]\circ[\![\alpha]\!](\theta,t) = [\![\alpha\ ;\ \beta]\!](\theta,t)\end{aligned}$$

- For $\alpha\,|\,\beta$, then $\theta(\alpha\,|\,\beta) = \theta(\alpha)\,|\,\theta(\beta)$ and using induction hypothesis on α and β

  $$\begin{aligned}[\![\theta(\alpha\,|\,\beta)]\!](\mathrm{id},t) &= [\![\theta(\alpha)\,|\,\theta(\beta)]\!](\mathrm{id},t) = [\![\theta(\alpha)]\!](\mathrm{id},t)\cup[\![\theta(\beta)]\!](\mathrm{id},t) = [\![\alpha]\!](\theta,t)\cup[\![\beta]\!](\theta,t)\\ &= [\![\alpha\,|\,\beta]\!](\theta,t)\end{aligned}$$

- For α^*, then $\theta(\alpha^*) = \theta(\alpha)^*$ and using induction hypothesis on α

  $$[\![\theta(\alpha^*)]\!](\mathrm{id},t) = [\![\theta(\alpha)^*]\!](\mathrm{id},t) = \bigcup_{n=0}^{\infty}[\![\theta(\alpha)]\!]^n(\mathrm{id},t) = \bigcup_{n=0}^{\infty}[\![\alpha]\!]^n(\theta,t) = [\![\alpha^*]\!](\theta,t)$$

- For $\alpha\ ?\ \beta\ :\ \gamma$, then $\theta(\alpha\ ?\ \beta\ :\ \gamma) = \theta(\alpha)\ ?\ \theta(\beta)\ :\ \theta(\gamma)$ and using induction hypothesis on α, β, γ

  $$\begin{aligned}[\![\alpha\ ?\ \beta\ :\ \gamma]\!](\mathrm{id},t) &= [\![\theta(\alpha)\ ?\ \theta(\beta)\ :\ \theta(\gamma)]\!](\mathrm{id},t) = [\![\theta(\beta)]\!]\circ[\![\theta(\alpha)]\!](\mathrm{id},t)\ \textbf{if}\ [\![\theta(\alpha)]\!](\mathrm{id},t)\ \textbf{else}\ [\![\theta(\gamma)]\!](\mathrm{id},t)\\ &= [\![\beta]\!]\circ[\![\alpha]\!](\theta,t)\ \textbf{if}\ [\![\alpha]\!](\theta,t)\ \textbf{else}\ [\![\gamma]\!](\theta,t) = [\![\alpha\ ?\ \beta\ :\ \gamma]\!](\theta,t)\end{aligned}$$

  $[\![\theta(\beta)]\!]\circ[\![\theta(\alpha)]\!](\mathrm{id},t) = [\![\beta]\!]\circ[\![\alpha]\!](\theta,t)$ had been proved above.

- For $rl\,[x_1 \texttt{<-}\, t_1,\ldots,x_n \texttt{<-}\, t_n]\,\{\alpha_1,\ldots,\alpha_m\}$ and using induction hypothesis for $\alpha_1,\ldots,\alpha_n$

  $$\begin{aligned}&[\![\theta(rl\,[x_1 \texttt{<-}\, t_1,\ldots,x_n \texttt{<-}\, t_n]\,\{\alpha_1,\ldots,\alpha_m\})]\!](\mathrm{id},t)\\ &= [\![\theta(rl\,[x_1 \texttt{<-}\, \theta(t_1),\ldots,x_n \texttt{<-}\, \theta(t_n)]\,\{\theta(\alpha_1),\ldots,\theta(\alpha_m)\})]\!](\mathrm{id},t)\\ &= \bigcup_{(rl,l,r,C)\in R} \mathsf{let}\ (\sigma,c) \leftarrowtail \mathsf{amcheck}(l,t,C,\mathrm{id}[x_1/\theta(t_1),\ldots,x_n/\theta(t_n)];\theta(\alpha_1)\cdots\theta(\alpha_m),\mathrm{id}) : \{c(\sigma(r))\}\\ &= \bigcup_{(rl,l,r,C)\in R} \mathsf{let}\ (\sigma,c) \leftarrowtail \mathsf{amcheck}(l,t,C,\mathrm{id}[x_1/\theta(t_1),\ldots,x_n/\theta(t_n)];\alpha_1\cdots\alpha_m,\theta) : \{c(\sigma(r))\}\\ &= [\![rl\,[x_1 \texttt{<-}\, t_1,\ldots,x_n \texttt{<-}\, t_n]\,\{\alpha_1,\ldots,\alpha_m\}]\!](\theta,t)\end{aligned}$$

  In the third equality we have used the mcheck identity twice: to introduce $\mathrm{id}[x_1/\theta(t_1),\ldots,x_n/\theta(t_n)]$ and then to extract it together with θ. Notice the identity only holds if $[\![\alpha_i]\!](\theta,t) = [\![\theta(\alpha_i)]\!](\mathrm{id},t)$ for all $1\leq i\leq m$, but this is true because of induction hypothesis.

- `matchrew` P `s.t` C `by` x_1 `using` α_1, $\dots, x_n$ `using` α_n and using induction hypothesis on $\alpha_1, \dots, \alpha_n$

$$\begin{aligned}
&[\![\theta(\texttt{matchrew } P \texttt{ s.t } C \texttt{ by } x_1 \texttt{ using } \alpha_1,\ \dots, x_n \texttt{ using } \alpha_n)]\!](\mathrm{id}, t)\\
&= [\![\texttt{matchrew } \theta_{-\{x_{k_j}\}}(P) \texttt{ s.t } \textstyle\bigwedge_j x_{k_j} := \theta(x_{k_j}) \wedge \theta(C) \texttt{ by } x_1 \texttt{ using } \theta(\alpha_1),\ \dots, x_n \texttt{ using } \theta(\alpha_n)]\!](\mathrm{id}, t)\\
&= \bigcup_{\sigma \in \Theta} \mathsf{let}\ t_1 \leftarrowtail [\![\theta(\alpha_1)]\!](\sigma, \sigma(x_1)), \dots, t_n \leftarrowtail [\![\theta(\alpha_n)]\!](\sigma, \sigma(x_n)) : \{\sigma[x_1/t_1, \dots, x_n/t_n](P)\}
\end{aligned}$$

where $\Theta = \mathsf{mcheck}(\theta_{-\{x_{k_j}\}}(P), t, \bigwedge_j x_{k_j} := \theta(x_{k_j}) \wedge \theta(C), \mathrm{id})$, and x_{k_j} are all variables designating subterms to be rewritten that are bound by θ. Since these variables are among $x_1, \dots, x_n$,

$$\sigma[x_1/t_1, \dots, x_n/t_n](P) = \sigma[x_1/t_1, \dots, x_n/t_n](\theta_{-\{x_{k_j}\}}(P)) = (\sigma \circ \theta)[x_1/t_1, \dots, x_n/t_n](P)$$

Using induction hypothesis twice, $[\![\theta(\alpha_i)]\!](\sigma, \sigma(x_i)) = [\![\alpha_i]\!](\sigma \circ \theta, \sigma(x_i))$. The proof would be finished if $\Theta \circ \theta = \mathsf{mcheck}(P, t, C, \theta)$.

$$\begin{aligned}
\Theta \circ \theta &= \mathsf{mcheck}(\theta_{-\{x_{k_j}\}}(P), t, \textstyle\bigwedge_j x_{k_j} := \theta(x_{k_j}) \wedge \theta(C), \mathrm{id}) \circ \theta_{-\{x_{k_j}\}}\\
&= \mathsf{mcheck}(\theta_{-\{x_{k_j}\}}(P), t, \theta_{-\{x_{k_j}\}}(\textstyle\bigwedge_j x_{k_j} := \theta(x_{k_j}) \wedge C), \mathrm{id}) \circ \theta_{-\{x_{k_j}\}}\\
&= \mathsf{mcheck}(P, t, \textstyle\bigwedge_j x_{k_j} := \theta(x_{k_j}) \wedge C, \theta_{-\{x_{k_j}\}})\\
&= \mathsf{mcheck}(P, t, C, \theta)
\end{aligned}$$

In the first line, θ can be replaced by itself with the variables x_{k_j} unbound, because the substitutions in Θ already have the x_{k_j} variables instantiated to $\theta(x_{k_j})$ in the condition. For the same reason, not replacing these variables in C is without effect, since they will be bound when C is recursively evaluated.

■

▸ **Proposition 17.** Given a rewrite theory with strategies $\mathcal{R} = (\Sigma, E, R; G, D)$, and $\alpha, \beta, \gamma \in \mathrm{Strat}_{\mathcal{R},\Omega}$

1. $[\![\texttt{idle} \mathbin{@} t]\!] = \{t\}$
2. $[\![\texttt{fail} \mathbin{@} t]\!] = \emptyset$
3. $[\![\alpha \,|\, \beta \mathbin{@} t]\!] = [\![\alpha \mathbin{@} t]\!] \cup [\![\beta \mathbin{@} t]\!]$
4. $[\![\alpha \,;\, \beta \mathbin{@} t]\!] = \mathsf{let}\ t' \leftarrowtail [\![\alpha \mathbin{@} t]\!] : [\![\beta \mathbin{@} t']\!]$
5. $[\![\alpha \ ?\ \beta : \gamma \mathbin{@} t]\!] = \begin{cases} [\![\alpha \,;\, \beta \mathbin{@} t]\!] & \text{si } [\![\alpha \mathbin{@} t]\!] \neq \emptyset \\ [\![\gamma \mathbin{@} t]\!] & \text{si } [\![\alpha \mathbin{@} t]\!] = \emptyset \end{cases}$
6. $[\![sl(t_1, \dots, t_n) \mathbin{@} t]\!] = \bigcup_{i=1}^{m_{sl}} \bigcup_{\theta \in \mathrm{vcheck}(p_{sl,i}, (t_1, \dots, t_n), C_{sl,i})} [\![\theta(\delta_{sl,\bar{c},i}) \mathbin{@} t]\!]$.

Items 1 to 5 are a direct translation using Proposition 16 of the semantic definition for the new semantics. Statement 6 comes from

$$\begin{aligned}
[\![sl(t_1, \dots, t_n) \mathbin{@} t]\!] = [\![sl(t_1, \dots, t_n)]\!](\mathrm{id}, t) = f_{sl}(t_1, \dots, t_n, t) = \bigcup d_i(\theta, t) &\overset{(C15)}{=} \bigcup [\![\delta_i]\!](\theta, t)\\
&\overset{(P16)}{=} \bigcup [\![\theta(\delta_i) \mathbin{@} t]\!]
\end{aligned}$$

where the union range as in the statement.

■

A.3 Proofs for the operational semantics

Now we start proving the equivalence between the previous denotational semantics in Section 3 and the operational semantics in Section 4. Then we deal with some properties of the operational semantics.

Our notion of equivalence is that for all terms t, t', strategy α, and substitution θ, $t \in [\![\alpha]\!](\theta, t)$ iff $t \mathbin{@} \alpha\, \theta \to^*_{s,c} t \mathbin{@} \varepsilon$. Since not all execution states have this form and in order to build an inductive proof, we ought to extend the semantic function to the whole $\mathcal{X}S_{\mathcal{R}}$. In the following, we will not explicitly mention the rewriting theory $\mathcal{R}$ we are dealing with, and will take for granted that $t, t', \dots$ are terms in $T_\Sigma(X)$; $\theta, \sigma, \dots$ are substitutions in VEnv; α, β, γ strategies; $q, q', \dots$ execution states; and s, s', s_0 stacks.

▸ **Definition.** We define the function $\mathrm{dsem} : \mathcal{XS}_{\mathcal{R}} \to \mathcal{P}_{\perp}(T_{\Sigma}(X))$ as follows

1. $\mathrm{dsem}(t \mathbin{@} \varepsilon) = \{t\}$
2. $\mathrm{dsem}(t \mathbin{@} \theta s) = \mathrm{dsem}(t \mathbin{@} s)$
3. $\mathrm{dsem}(t \mathbin{@} \alpha s) = \mathsf{let}\ t' \leftarrowtail [\![\alpha]\!](\mathrm{vctx}(s), t) : \mathrm{dsem}(t' \mathbin{@} s)$
4. $\mathrm{dsem}(\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} s) = \mathsf{let}\ \overset{n}{\underset{i=1}{}}\ t_i \leftarrowtail \mathrm{dsem}(q_i) : \mathrm{dsem}(t[x_i/t_i]_{i=1}^n \mathbin{@} s)$
5. $\mathrm{dsem}(\mathrm{rewc}(p : q, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s) =$
$$\mathsf{let}\ t' \leftarrowtail \mathrm{dsem}(q) : \bigcup_{\sigma_m \in \mathsf{match}(p,t')} \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta_s) : \{m(\sigma'(r))\}$$

This function is a generalization of $[\![\cdot]\!]$ because $\mathrm{dsem}(t \mathbin{@} \alpha\, \theta) = [\![\alpha]\!](\theta, t)$ (as easily follows from the definition).

▸ **Proposition 18.** For $\alpha, \beta \in \mathrm{Strat}_{\mathcal{R},\Omega}$ and $s \in \mathrm{Stack}$

1. $\mathrm{dsem}(t \mathbin{@} \alpha\ ;\ \beta\, s) = \mathrm{dsem}(t \mathbin{@} \alpha\beta s)$
2. $\mathrm{dsem}(t \mathbin{@} \alpha \mid \beta\, s) = \mathrm{dsem}(t \mathbin{@} \alpha s) \cup \mathrm{dsem}(t \mathbin{@} \beta s)$
3. $\mathrm{dsem}(\mathrm{rewc}(p : t \mathbin{@} \alpha\, \theta_s, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s) = \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(t' \mathsf{\ =>\ } p \wedge C, \sigma; \alpha\bar{\alpha}, \theta_s) : \{m(\sigma'(r))\}$ for t' such that $\sigma(t') = t$.

A routine calculation proves the first statement (we write θ for $\mathrm{vctx}(s)$)

$$\begin{aligned}
\mathrm{dsem}(t \mathbin{@} \alpha\ ;\ \beta\, s) &= \mathsf{let}\ t' \leftarrowtail [\![\alpha\ ;\ \beta]\!](\theta, t) : \mathrm{dsem}(t' \mathbin{@} s) \\
&= \mathsf{let}\ t' \leftarrowtail (\mathsf{let}\ t_m \leftarrowtail [\![\alpha]\!](\theta, t) : [\![\beta]\!](\theta, t_m)) : \mathrm{dsem}(t' \mathbin{@} s) \\
&= \mathsf{let}\ t_m \leftarrowtail [\![\alpha]\!](\theta, t) : (\mathsf{let}\ t' \leftarrowtail [\![\beta]\!](\theta, t_m) : \mathrm{dsem}(t' \mathbin{@} s)) \\
&= \mathsf{let}\ t_m \leftarrowtail [\![\alpha]\!](\theta, t) : \mathrm{dsem}(t_m \mathbin{@} \beta\, s)) \\
&= \mathrm{dsem}(t \mathbin{@} \alpha\beta\, s)
\end{aligned}$$

and the second

$$\begin{aligned}
\mathrm{dsem}(t \mathbin{@} \alpha \mid \beta\, s) &= \mathsf{let}\ t' \leftarrowtail [\![\alpha \mid \beta]\!](\theta, t) : \mathrm{dsem}(t' \mathbin{@} s) \\
&= \mathsf{let}\ t' \leftarrowtail [\![\alpha]\!](\theta, t) \cup [\![\beta]\!](\theta, t) : \mathrm{dsem}(t' \mathbin{@} s) \\
&= \mathsf{let}\ t' \leftarrowtail [\![\alpha]\!](\theta, t) : \mathrm{dsem}(t' \mathbin{@} s) \cup \mathsf{let}\ t' \leftarrowtail [\![\beta]\!](\theta, t) : \mathrm{dsem}(t' \mathbin{@} s) \\
&= \mathrm{dsem}(t' \mathbin{@} \alpha\, s) \cup \mathrm{dsem}(t' \mathbin{@} \beta\, s)
\end{aligned}$$

For the third statement,

$$\begin{aligned}
&\mathrm{dsem}(\mathrm{rewc}(p : t \mathbin{@} \alpha\theta_s, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s) \\
&\quad = \mathsf{let}\ t_m \leftarrowtail [\![\alpha]\!](\theta_s, t) : \bigcup_{\sigma_m \in \mathsf{match}(p,t_m)} \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta_s) : \{m(\sigma'(r))\} \\
&\quad = \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(t' \mathsf{\ =>\ } p \wedge C, \sigma; \alpha\bar{\alpha}, \theta_s) : \{m(\sigma'(r))\}
\end{aligned}$$

by the definition of check.

■

▸ **Proposition 19.** The execution tree for $\rightarrow_{s,c}$ is finitary.

In other words, the number of direct successors of any state $q \in \mathcal{XS}_{\mathcal{R}}$ for $\rightarrow_{s,c}$ is finite. Terms like $t \mathbin{@} \theta s$ have $t \mathbin{@} s$ as the only successor. Looking at Figures 2 and 3, it is easy to conclude that the successors of $t \mathbin{@} \alpha s$ are finitely many (no more that two reductions except for `matchrew` and rule application, which have a successor for each possible match, anyhow a finite number).

Reasoning inductively, structured states have a finite number of successors too. In the case of rewc, they are the replacement of the substate by its successors (a finite number by induction hypothesis) or the rewc

term for the next rewriting fragment or the plain term $t \mathbin{@} s$ after successful evaluation of the condition. In the last two cases the successor are a finite amount because matches are finitely many. In the case of subterm, the states that may follow are the replaced term $t[x_i/t_i] \mathbin{@} s$ upon successful termination, and the evolution of the subterms. In the last case the number of possibilities is the sum of the possibilities for all of its subterms, a finite number by induction hypothesis.

■

On the contrary, note that the execution tree for $\twoheadrightarrow$ may not be finitary. Suppose two rules are defined for integer numbers: $n \Rightarrow s(n)$ labeled *inc* and $n \Rightarrow -m$ **if** $n \to m$ labeled *neg*. The execution state $0 \mathbin{@} neg\{inc^*\}$ has $-n \mathbin{@} \varepsilon$ as successor for all $n \in \mathbb{N}$. In terms of $\to_{s,c}$ there is an infinite execution

$$0 \mathbin{@} neg\{inc^*\} \to_c ... \to_c \mathrm{rewc}(m : k \mathbin{@} inc^*, [n \mapsto 0], \mathtt{true}, \varepsilon, \mathrm{id}, -m; 0) \mathbin{@} \varepsilon \to_c ...$$

▸ **Corollary 20.** The execution tree for $\to_{s,c}$ is infinite iff it contains an infinite execution.

The *if* case is obvious. The *only if* case is also easy: the tree is finitary, so to be infinite, by Kőnig lemma [6], it must have an infinite path (execution).

■

▸ **Definition.** The *call depth* cdepth of a state $q \in \mathcal{X}S_{\mathcal{R}}$ is defined as:

1. For $t \mathbin{@} s$, the number of substitutions in s.
2. For $\mathrm{subterm}(x_1 : q_1, ..., x_n : q_n; t) \mathbin{@} s$, the number of substitutions in s plus the maximum of zero and $\mathrm{cdepth}(q_i) - 1$ for all i.
3. For $\mathrm{rewc}(p : q, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s$, the number of variable environments in s plus the maximum of zero and $\mathrm{cdepth}(q) - 1$.

The *call depth* of an execution $q_1 \to_{s,c} q_2 \to^*_{s,c} q_n \to^*_{s,c} ...$ of length $1 \leq l \leq \infty$ is

$$\sup \{ \mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_n) \mid 1 \leq n \leq l, n \leq m \leq l \}$$

or the maximum call depth of the executions in the proofs for the [else] rules if this is greater. It can be infinity.

The call depth is a measure of the number of nested calls, like the call stack size in a modern computer. One is subtracted to the call depths of the substates in items 2 and 3 above because the substitutions at their bottoms never come from a strategy call but from the matching environment of a `matchrew`, or the replication of the outer context in rewc. The call depth of a state increases when more context is added to the bottom of its queue, but the call depth of an execution is indifferent when all their states stacks are extended the same way. Another property is that the call depth of the suffixes of any execution are always less or equal than the depth of the whole.

For convenience and abusing notation, we write $q \mathbin{@} s$ to represent a state in any of its forms with a global stack s. That is, $q \mathbin{@} s$ can be $t \mathbin{@} s$, $\mathrm{subterm}(...) \mathbin{@} s$ or $\mathrm{rewc}(...) \mathbin{@} s$.

▸ **Lemma 21.**

1. If $t \mathbin{@} s_1\, \theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and $t_m \mathbin{@} s_2 \to^*_{s,c} q$ then $t \mathbin{@} s_1 s_2 \to^*_{s,c} q$ where $\theta = \mathrm{vctx}(s_2)$.

   Moreover, the length of the resulting execution is the sum of the lengths of the original ones, and its call depth is the maximum.
2. If $q_0 \mathbin{@} s_1 s_2 \to^*_{s,c} q'$ and $s_1 \neq \varepsilon$ then

$$q' \in \{q \mathbin{@} s' s_2 : q_0 \mathbin{@} s_1 \theta \to^*_{s,c} q \mathbin{@} s'\} \cup \{q : \exists\, t_m \quad q_0 \mathbin{@} s_1 \theta \to^*_{s,c} t_m \mathbin{@} \varepsilon \ \wedge\ t_m \mathbin{@} s_2 \to^*_{s,c} q\}$$

3. If $t \mathbin{@} s_1 s_2 \to^*_{s,c} t' \mathbin{@} \varepsilon$ then there is a term t_m such that $t \mathbin{@} s_1 \theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and $t_m \mathbin{@} s_2 \to^*_{s,c} t' \mathbin{@} \varepsilon$.

Some common facts follow easily from the inspection of the rules and axioms of the semantics definition:

- The global stack s in $q \mathbin{@} s$ only changes by a $\to_{s,c}$ reduction if $q = t \mathbin{@} s$ for some term t.

- Only the top of the stack can be popped by a reduction, $t \mathbin{@} \alpha\, s \to_{s,c} q \mathbin{@} s_\alpha\, s$ for some stack s_α.
- Reductions only depend on the top of the stack and the variable environment. Thus, the stack can be extended from below without effect if the latter is preserved, i.e. $q \mathbin{@} s\, \theta \to_{s,c} q' \mathbin{@} s'$ implies $q \mathbin{@} s\, s_0 \to_{s,c} q \mathbin{@} s' s_0$ where $\theta = \mathrm{vctx}(s_0)$, which may be omitted if it is id.

Using these basic facts, we prove the statements:

1. Replace θ by s_2 in the stacks' bottoms of the first execution. Then $t \mathbin{@} s_1 s_2 \to^*_{s,c} t_m \mathbin{@} s_2$ is obtained, and joining it with the second execution, the statement holds.

   While it is clear that the length is the sum of the lengths of the components, we will discuss the call depth of the combined execution. It is the following maximum (the sup is a max because both executions are finite)

   $$\max \{\ \mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_n) \mid 1 \le n \le l, n \le m \le l\ \}$$

   Let l_1 be the length of the first execution. For $n > l_1$, i.e. for states to the right of $t_m \mathbin{@} s_2$, the numbers $\mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_n)$ are the same as in the original execution because the states are exactly the same. The states before $t_m \mathbin{@} s_2$ have been extended with s_2 at the bottom of their stacks, so their call depth is greater than that of $t_m \mathbin{@} s_2$. Hence, terms like $\mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_n)$ with $n < l_1$ and $m \ge l_1$ can be ignored when taking the maximum because they are bounded above by $\mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_{l_1})$. The only missing case is $\mathrm{cdepth}(q_m) - \mathrm{cdepth}(q_n)$ with both $n, m < l_1$ but they are at most (and reach) the call depth of the first execution, because they are the states of this execution extended uniformly at the bottom.

2. The proof is carried out by induction on the length k of the derivation $q \mathbin{@} s_1 s_2 \to^*_{s,c} q'$

   - Base case ($k = 0$): then $q_0 \mathbin{@} s_1 s_2 = q'$, so q' is in the first set with the 0-length execution and $s' = s_1$.
   - Inductive case: we have $q_0 \mathbin{@} s_1 s_2 \to_{s,c} q_1 \to^k_{s,c} q'$. If q_0 is a subterm or rewc state, the stack remains unchanged in the first step, so we can apply induction hypothesis on $q_1 \mathbin{@} s_1 s_2 \to^k_{s,c} q'$ to conclude. Otherwise, $q_0 \mathbin{@} s_1 s_2 = t \mathbin{@} s_1 s_2$ for some term t. And since the stack s_1 is not empty, either $s_1 = \sigma s_1'$ or $s_1 = \alpha\, s_1'$.

     In the first case, $t \mathbin{@} \sigma s_1' s_2 \to_c t \mathbin{@} s_1' s_2$. If $s_1' = \varepsilon$ then q' is in the second set with $t_m = t$. Otherwise, we apply induction hypothesis to the rest of the derivation $t \mathbin{@} s_1' s_2 \to^k_{s,c} q'$. Here, we have two possibilities:
     - There is a term t_m such that $t \mathbin{@} s_1' s_2 \to^*_{s,c} t_m \mathbin{@} s_2$ and $t_m \mathbin{@} s_2 \to^*_{s,c} q'$. Hence, our q' is in the second set with $t_m = t$. This follows from the second execution above and $t \mathbin{@} \sigma s_1 \theta \to_c t \mathbin{@} s_1 \theta \to_c t \mathbin{@} s_1' \theta \to^*_{s,c} t_m \mathbin{@} \theta \to_c t_m \mathbin{@} \varepsilon$.
     - $t \mathbin{@} s_1' \theta \to^*_{s,c} q \mathbin{@} s'$ for some state q and $q' = q \mathbin{@} s' s_2$. Thus, q' is in the first set with $t \mathbin{@} s_1 \theta = t \mathbin{@} \sigma s_1'\, \theta \to_c t \mathbin{@} s_1'\, \theta \to^*_{s,c} q \mathbin{@} s'$.

     The second case, $s_1 = \alpha s_1'$, whose first step is $t \mathbin{@} \alpha s_1' s_2 \to_{s,c} q_1 \mathbin{@} s_\alpha s_1' s_2$ is almost identical.

3. We only have to apply (2) with $q_0 \mathbin{@} s_1 s_2 = t \mathbin{@} s_1 s_2$ and $q' = t' \mathbin{@} \varepsilon$ if $s_1 \ne \varepsilon$ (otherwise is trivial).

■

▸ **Lemma 22.** Given $f : \mathrm{VEnv} \times T_\Sigma(X) \to \mathcal{P}_\perp(T_\Sigma(X))$, assume

$$\forall\, t, t' \in T_\Sigma(X) \quad \forall \theta \in \mathrm{VEnv} \qquad t' \in f(\theta, t) \implies t \mathbin{@} \alpha\, \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$$

For any $k \in \mathbb{N}$, if $t' \in f^k(\theta, t)$ then $t \mathbin{@} \alpha^* \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$.

By induction on k. If $k = 0$ (the base case), $f^0(\theta, t) = \{t\}$ so $t = t'$ and we are done using the rule $t \mathbin{@} \alpha^* \theta \to_c t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$. In the inductive case, assume the lemma holds for k. We want to prove that if $t' \in f^{k+1}(\theta, t)$ then $t \mathbin{@} \alpha^* \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$. The definition of composition and $f^{k+1} = f \circ f^k$ imply that there is a term t_m such that $t_m \in f^k(\theta, t)$ and $t' \in f(\theta, t_m)$. Using, in this order, the second rule for the iteration, the statement assumption, and the induction hypothesis, we conclude

$$t \mathbin{@} \alpha^* \theta \to_c t \mathbin{@} \alpha\alpha^* \theta \to^*_{s,c} t_m \mathbin{@} \alpha^* \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$$

■

▸ **Lemma 23.** Given $f : \mathrm{VEnv} \times T_\Sigma(X) \to \mathcal{P}_\perp(T_\Sigma(X))$, assume

$$\forall\, t, t' \in T_\Sigma(X) \quad \forall \theta \in \mathrm{VEnv} \qquad t \mathbin{@} \alpha\, \theta \to^*_{s,c} t' \mathbin{@} \varepsilon \implies t' \in f(\theta, t)$$

If $t \mathbin{@} \alpha^* \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$ and the number of states in the execution with stack $\alpha^* \theta$ is exactly $k+1$, then $t' \in f^k(\theta, t)$.

By induction on k. If $k = 0$, the only allowed execution is $t \mathbin{@} \alpha^* \theta \to_c t \mathbin{@} \theta \to_c t \mathbin{@} \theta$. So $t = t'$ and $t \in f^0(\theta, t) = \{t\}$. If $k > 0$, we can decompose the execution in two parts: before the penultimate state with stack $\alpha^* \theta$ and after this state. The first part is $t \mathbin{@} \alpha^* \theta \to^*_{s,c} t_m \mathbin{@} \alpha^* \theta$ and it can be completed with $t_m \mathbin{@} \alpha^* \theta \to_c t_m \mathbin{@} \theta \to_c t_m \mathbin{@} \varepsilon$. It contains exactly k states with $\alpha^* \theta$, so by induction hypothesis, $t_m \in f^{k-1}(\theta, t)$. The second execution is $t_m \mathbin{@} \alpha^* \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$, but its second state must be $t_m \mathbin{@} \alpha\alpha^* \theta$ and the last states need be $t' \mathbin{@} \alpha^* \theta$ and $t' \mathbin{@} \theta$. Forgetting about the first and last two states and removing α^* from the stacks, we get $t_m \mathbin{@} \alpha\, \theta \to^*_{s,c} t' \mathbin{@} \varepsilon$. By the hypothesis of the lemma, $t \in f(\theta, t_m)$. Combining both results, $t' \in f(\theta, t_m)$ and $t_m \in f^{k-1}(\theta, t)$, we conclude $t' \in f^k(\theta, t)$ as we wanted.

■

▸ **Lemma 24.** Given $f : \mathrm{VEnv} \times T_\Sigma(X) \to \mathcal{P}_\perp(T_\Sigma(X))$ and assuming $\perp \in f^k(\theta, t)$ for some $k \in \mathbb{N}$, there is a term $t_\perp$ and $k_\perp \le k$ such that $t_\perp \in f^{k_\perp}(\theta, t)$ and $\perp \in f(\theta, t_\perp)$.

The proof is by induction on k. For $k = 0$, the premises are never satisfied, since $f^0(\theta, t) = \{t\}$. The statement clearly holds for $k = 1$ with $t_\perp = t$ and $k_\perp = 0$. For $k \ge 2$, remember $f^k(\theta, t) = \mathsf{let}\ t_m \leftarrowtail f^{k-1}(\theta, t) : f(\theta, t_m)$. The following situations can happen:

- $\perp \in f(\theta, t_m)$. Then $t_\perp = t_m$ and $k_\perp = k - 1$.
- $\perp \in f^{k-1}(\theta, t)$. Then the induction hypothesis proves the lemma.

■

We define $\mathrm{len}(q)$ for $q \in \mathcal{XS}_\mathcal{R}$ as the maximum length of a $\to_{s,c}$ execution from q.

▸ **Lemma 25.**

1. $\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} \mathrm{subterm}(x_1 : q'_1, \ldots, x_n : q'_n; t) \mathbin{@} \varepsilon$ if $q_i \to^*_{s,c} q'_i$ for all i.
2. If $\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} \mathrm{subterm}(x_1 : q'_1, \ldots, x_n : q'_n; t) \mathbin{@} \varepsilon$ then $q_i \to^*_{s,c} q'_i$ for all i.
3. If $q_i \to^*_{s,c} t_i \mathbin{@} \varepsilon$ for all $1 \le i \le n$ then $\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$.
4. If $\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} t' \mathbin{@} \varepsilon$ then there are terms $t_1, \ldots, t_n$ such that $q_i \to^*_{s,c} t_i \mathbin{@} \varepsilon$ for all $1 \le i \le n$ and $t' = t[x_i/t_i]_{i=1}^n$.
5. $\mathrm{len}(\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon) = \sum_{k=1}^n \mathrm{len}(q_k) + 1$.

Statements (3) and (4) are a corollary of statements (1) and (2). Induction is used to prove the latter.

1. By induction on the execution lengths from q_i. Suppose all lengths are 0 then $q_i = q'_i$ and the empty execution solves the statement. Otherwise, there is at least a positive number in the n-tuple of execution lengths. We assume the statement is true for any collection of derivations of lower or equal length with at least one strictly lower coordinate. Suppose, without loss of generality, that $q_1 \to_{s,c} q_{1,1} \to^*_{s,c} q'_1$. Then by $[\mathrm{prl}_s]$ or $[\mathrm{prl}_c]$ we can assert

   $$\mathrm{subterm}(x_1 : q_1, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to_{s,c} \mathrm{subterm}(x_1 : q_{1,1}, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon$$

   And the remaining execution to be applied in x_1 is strictly shorter than the original one. The derivation can be completed by induction hypothesis.

2. Now, induction is based on the length of the execution in the premise. Suppose the length is 0, then $q_i = q'_i$ and the empty executions from every q_i make the statement true. If the length is positive, there must be an initial $[\mathrm{prl}_c]$ or $[\mathrm{prl}_s]$ transition

   $$\mathrm{subterm}(x_1 : q_1, \ldots, x_i : q_i, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon \to_{s,c} \mathrm{subterm}(x_1, q_1, \ldots, x_i : q_{i,1}, \ldots, x_n : q_n; t) \mathbin{@} \varepsilon$$

   and for this to be true $q_i \to_{s,c} q_{i,1}$ must hold. By induction hypothesis on the execution from the right-hand side, $q_j \to^*_{s,c} q'_j$ for all $j \neq i$ and $q_i \to_{s,c} q_{i,1} \to^*_{s,c} q'_i$. So we have found the desired executions.

3. It is enough to apply (1) and then subterm$(x_1 : t_1 \mathbin{@} \varepsilon, \dots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} \varepsilon \to_c t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$.
4. $t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$ must be preceded by subterm$(x_1 : t_1 \mathbin{@} \varepsilon, \dots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} \varepsilon$, so we can apply (2) to conclude.
5. For the last statement, we observe in the proofs of the previous items that the size of the subterm execution we have built in (1,3) is the sum of the lengths of their subexecutions plus the ending transition. And the length of the substate executions we have recovered in (2,4) sum the length of the original derivation except for the last $\to_c$ transition. Then we conclude respectively the $\geq$ and the $\leq$ of the fifth statement, so we can assert the identity.

■

▸ **Lemma 26.** For any conditions C_1 and C_2, and $\bar{\alpha}_1 \in \mathrm{Strat}^*_{\mathcal{R},\Omega}$ for the rewriting fragments of C_1 and $\bar{\alpha}_2$ for C_2

$$\mathsf{check}(C_1 \wedge C_2, \sigma, \bar{\alpha}_1\bar{\alpha}_2, \theta) = \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(C_1, \sigma, \bar{\alpha}_1, \theta) : \mathsf{check}(C_2, \sigma', \bar{\alpha}_2, \theta)$$

and if C_1 is equational ($\bar{\alpha}_1 = \varepsilon$), the let is a normal union.

Follows from the recursive definition of check.

■

▸ **Theorem 27.** Forall $t \in T_\Sigma(X)$ and $q \in \mathcal{X}S_\mathcal{R}$

$$t \in \mathrm{dsem}(q) \quad \iff \quad q \to^*_{s,c} t \mathbin{@} \varepsilon$$

and $\bot \in \mathrm{dsem}(q)$ iff there is an infinite execution from q.

The simultaneous proof of both $\Rightarrow$ and $\Leftarrow$ will be done by induction, on the structure of q but also on the approximants of the semantic function. Remember that the meaning of the strategy definitions in a module is given by functions d_i, and d_i is defined as the i-th component of the fixed point $\mathsf{FIX}\, F$ of the function $F(d_1, \dots, d_{|D|}) = (\llbracket \delta_1 \rrbracket, \dots, \llbracket \delta_{|D|} \rrbracket)$ where δ_i is the strategy definition term. By Theorem 14 we know that $\mathsf{FIX}\, F = \sup\{F^n(\{\bot\}, \dots, \{\bot\}) : n \in \mathbb{N}\}$, where the supremum (see Proposition 2 and Proposition 11) is the union (perhaps removing $\bot$) by coordinates. We write $d_k^{(n)} = F_k^n(\{\bot\}, \dots, \{\bot\})$. And replacing d_k by $d_k^{(n)}$, we also define $f_{sl}^{(n)}$, $\llbracket \alpha \rrbracket^{(n)}$, $\mathrm{dsem}^{(n)}$ and $\mathsf{check}^{(n)}$. Note that

$$d_k^{(n+1)} = F_k(d_1^{(n)}, \dots, d_{|D|}^{(n)}) = \llbracket \delta_k \rrbracket (d_1^{(n)}, \dots, d_{|D|}^{(n)})$$

so $d_k^{(n)} \sqsubseteq d_k^{(n+1)}$ because F is monotonic. The other functions are monotonic too.

Induction property. The induction property in $\mathbb{N} \times \mathcal{X}S_\mathcal{R}$ with the order we detail below is $\mathbf{p}(n, q) \iff$

1. Forall $1 \leq k \leq |D|$, $t \in d_k^{(n)}(\theta, t_0)$ implies $t_0 \mathbin{@} \delta_k\theta \to^*_{s,c} t \mathbin{@} \varepsilon$.
2. Forall $t \in \mathrm{dsem}^{(n)}(q)$, $q \to^*_{s,c} t \mathbin{@} \varepsilon$.
3. If $q \to^*_{s,c} t \mathbin{@} \varepsilon$ with call depth at most n, then $t \in \mathrm{dsem}^{(n)}(q)$.
4. If $\bot \in \mathrm{dsem}^{(n)}(q)$, then there is a $\to_{s,c}$ execution whose call depth is at least $n+1$ ignoring [else] proofs or containing infinitely many consecutive iterations of the same strategy.
5. If $\bot \notin \mathrm{dsem}^{(n)}(q)$, then all executions from q are finite and its call depth is at most n.

The induction order is the lexicographic order for the well-defined $\mathbb{N}^4$ tuple

$$(n, \text{strategy constructors in the whole } q, \text{nested } \mathcal{X}S_\mathcal{R} \text{ constructors in } q, \text{condition fragments in } q)$$

It is a well-founded order, and every syntactic substate of an execution state is below the whole state.

p implies the theorem. Assume $\mathbf{p}(n,q)$ for every $n \in \mathbb{N}$ and $q \in \mathcal{XS}_{\mathcal{R}}$. Given q, if $t \in \mathrm{dsem}(q)$, by the fixed-point definition recalled above, $t \in \mathrm{dsem}^{(n_0)}(q)$ for some $n_0 \in \mathbb{N}$. From $\mathbf{p}(n_0,q)$ we conclude $\Rightarrow$ by item (2). If we are given a derivation $q \to^*_{s,c} t \mathbin{@} \varepsilon$ and n_0 is its call depth, $\mathbf{p}(n_0,q)$ lets us conclude $\Leftarrow$ by item (3). If $\bot \notin \mathrm{dsem}(q)$ then $\bot \notin \mathrm{dsem}^{(n_0)}(q)$ for some $n_0 \in \mathbb{N}$ (by a simple generalization of Proposition 7), then all executions for q are finite. If otherwise $\bot \in \mathrm{dsem}(q)$, then $\bot \in \mathrm{dsem}^{(n)}(q)$ for all $n \in \mathbb{N}$, so an iteration is executed infinitely many consecutive times from q or there are derivations of arbitrary big call depth and hence length by (4). In either case, there is an infinite execution, either directly or by Corollary 20.

Induction for 1. First, we will prove (1) for any $q \in \mathcal{XS}_{\mathcal{R}}$.

- For $n = 0$, $d_k^{(0)} = \{\bot\}$ (the constant function) so $t \in d_k^{(0)}(\theta, t_0)$ never holds and (1) is true vacuously.
- Now assume $\mathbf{p}(n,q)$ for all $q \in \mathcal{XS}_{\mathcal{R}}$ to prove the first statement of $\mathbf{p}(n+1,q)$. If $t \in d_k^{(n+1)}(\theta, t_0)$, and by the recursive definition of the approximants, $t \in [\![\delta_i]\!]^{(n)}(\theta, t_0) = \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \delta_i\, \theta)$. The induction hypothesis (item 2) let us conclude $t_0 \mathbin{@} \delta_i\, \theta \to^*_{s,c} t \mathbin{@} \varepsilon$.

Induction for 2-5:. now we will prove the rest of the statements ranging on q for any $n \in \mathbb{N}$. First, we will treat the stack cases. We will usually make explicit which elements are in the semantics $\mathrm{dsem}^{(n)}$ and which derivation can follow from the state, so that the properties follow easily. The base cases are:

- $\boxed{\texttt{idle}}$ From one side $\mathrm{dsem}^{(n)}(t_0 \mathbin{@} \texttt{idle}\, \theta) = [\![\texttt{idle}]\!]^{(n)}(\theta, t_0) = [\![\texttt{idle}]\!](\theta, t_0) = \{t_0\}$, and from the other side, $t_0 \mathbin{@} \texttt{idle}\, \theta \to_c t_0 \mathbin{@} \theta \to_c t_0 \mathbin{@} \varepsilon$ is the only allowed execution from $t_0 \mathbin{@} \texttt{idle}\, \theta$. It is clear that (2), (3) and (5) hold. Statement (4) holds trivially.
- $\boxed{\texttt{fail}}$ We know $\mathrm{dsem}^{(n)}(t_0 \mathbin{@} \texttt{fail}\, \theta) = [\![\texttt{fail}]\!]^{(n)}(\theta, t_0) = [\![\texttt{fail}]\!](\theta, t_0) = \emptyset$ and that no $\to_{s,c}$ transition starts from $t_0 \mathbin{@} \texttt{fail}\, \theta$. So all statements are satisfied vacuously.
- $\boxed{\texttt{match}\ P\ \texttt{s.t}\ C}$ Again

  $$\mathrm{dsem}^{(n)}(t_0 \mathbin{@} \texttt{match}\ P\ \texttt{s.t}\ C\, \theta) = [\![\texttt{match}\ P\ \texttt{s.t}\ C]\!]^{(n)}(\theta, t_0) = [\![\texttt{match}\ P\ \texttt{s.t}\ C]\!](\theta, t_0)$$

  which equals $\emptyset$ if $\mathsf{mcheck}(P, t, \theta) \neq \emptyset$ or $\{t_0\}$ otherwise. From the other side,

  $$t_0 \mathbin{@} \texttt{match}\ P\ \texttt{s.t}\ C\, \theta \to_c t_0 \mathbin{@} \theta \to_c t_0 \mathbin{@} \varepsilon$$

  if the matching is not empty, and no transition leaves the state otherwise. All this implies (2), (3) and (5), while (4) is trivially true.
- $\boxed{rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]}$ Now

  $$\begin{aligned}\mathrm{dsem}^{(n)}(t_0 \mathbin{@} rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]\ \theta) &= [\![rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]]\!]^{(n)}(\theta, t_0) \\ &= [\![rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]]\!](\theta, t_0) \\ &= \mathsf{ruleApply}(t_0, rl, \mathrm{id}[x_i/\theta(t_i)]_{i=1}^n)\end{aligned}$$

  and iff t belongs to such a set we have

  $$t_0 \mathbin{@} rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]\ \theta \to_c t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$$

  This implies (2), (3) and (5). The premise of (4) cannot happen.

And now the inductive cases. We will sometimes resort to Proposition 18 and understand that it is extended to the superscripted dsem because the proof is exactly the same.

- $\boxed{rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]\,\{\alpha_1, \ldots, \alpha_k\}}$ In this case, only reducing with the second rule of Figure 3 is allowed.

  $$t \mathbin{@} rl\,[x_1 \texttt{<-} t_1, \ldots, X_m \texttt{<-} t_m]\,\{\alpha_1, \ldots, \alpha_k\}\, \theta \to_c \mathrm{rewc}(r_1 : \sigma(l_1) \mathbin{@} \alpha_1\theta, \sigma, C, \alpha_2 \cdots \alpha_k, r, c; t) \mathbin{@} \theta$$

  for any rule $(rl, l, r, C_0 \wedge l_1 \texttt{ => } r_1 \wedge C)$ and match $(\sigma, c) \in \mathsf{amatch}(l, t, \mathrm{id}[x_i/t_i]_{i=1}^m, C_0)$. The rank decreases in the transition (a strategy constructor less), so we can apply induction hypothesis on the right-hand side. By Proposition 18 the $\mathrm{dsem}^{(n)}$ of both sides is the same. Properties (2), (3), (4), and (5) follow immediately.

- $\boxed{\theta\, s}$ The only allowed reduction here is $t_0 \mathbin{@} \theta s \to_c t_0 \mathbin{@} s$ and, by definition of dsem, the semantics of both sides are exactly the same. Properties (2) and (5) follow directly.

  (3). Given a derivation $t_0 \mathbin{@} \theta s \to_c t_0 \mathbin{@} s \to^*_{s,c} t \mathbin{@} \varepsilon$ whose call depth is at most n, the suffix derivation from $t_0 \mathbin{@} s$ has lower or equal call depth. So the induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} s)$ can be applied to conclude that $t \in \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \theta\, s)$.

  (4). If $\bot$ is in the $\mathrm{dsem}^{(n)}$, there is an execution from $t_0 \mathbin{@} s$ with call depth at least $n+1$, so the full execution has call depth at least $n+1$ because it must be greater or equal than that of its suffix.

- $\boxed{\alpha\, s}$ where s contains at least a strategy expression. We know

$$R := \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \alpha\, s) = \mathsf{let}\ t_m \leftarrowtail [\![\alpha]\!]^{(n)}(\theta, t_0) : \mathrm{dsem}(t_m \mathbin{@} s)$$

  and we can invoke the induction hypotheses $\mathbf{p}(n, t_0 \mathbin{@} \alpha\, \theta)$ and $\mathbf{p}(n, t_m \mathbin{@} s)$ where $\theta = \mathrm{vctx}(s)$ (both are lower in the order because the number of strategy constructors decreases).

  (2). From the induction hypotheses, $t_0 \mathbin{@} \alpha\, \theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and $t_m \mathbin{@} s \to^*_{s,c} t \mathbin{@} \varepsilon$ if $t \in \mathrm{dsem}(t_0 \mathbin{@} \alpha\, s)$. These executions can be joined by Lemma 21 (item 1) to build the desired one.

  (3). We are give an execution $t_0 \mathbin{@} \alpha\, s \to^*_{s,c} t \mathbin{@} \varepsilon$. This execution can be split using the third statement of Lemma 21 in two executions $t_0 \mathbin{@} \alpha\, \theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and $t_m \mathbin{@} s \to^*_{s,c} t \mathbin{@} \varepsilon$ for some term t_m (both clearly have lower or equal call depth than the original one). The induction hypotheses say that $t_m \in \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \alpha\, \theta) = [\![\alpha]\!]^{(n)}(\theta, t_m)$ and $t \in \mathrm{dsem}(t_m \mathbin{@} s)$, so we can conclude $t \in \mathrm{dsem}(t_0 \mathbin{@} \alpha s)$.

  (4). $\bot \in R$ if $\bot \in [\![\alpha]\!]^{(n)}(\theta, t_0)$ or $\bot \in \mathrm{dsem}(t_m \mathbin{@} s)$. By induction hypothesis and in any case, the combined execution's call depth is at least $n+1$ because one of its components is.

  (5). If $\bot \notin R$ then $\bot$ is neither in $\mathrm{dsem}^{(n)}(t_m \mathbin{@} s)$ for any t_m in $[\![\alpha]\!]^{(n)}(\theta, t_0)$ nor in $[\![\alpha]\!]^{(n)}(\theta, t_0)$ itself, so all executions from these states are finite and their call depth is at most n by Lemma 21 (1).

- $\boxed{\alpha\, ;\, \beta}$ By Proposition 18, $t_0 \mathbin{@} \alpha\, ;\, \beta\, \theta$ and $t_0 \mathbin{@} \alpha\beta\, \theta$ are semantically equivalent, $t_0 \mathbin{@} \alpha\, ;\, \beta\, \theta \to_c t_0 \mathbin{@} \alpha\beta\, \theta$ is the single possible reduction from the state, and $t_0 \mathbin{@} \alpha\beta\, \theta$ is lower than $t_0 \mathbin{@} \alpha\, ;\, \beta\, \theta$ (the sequence constructor is removed). So this case is reduced to the previous one, using the induction hypothesis.

- $\boxed{\alpha \,|\, \beta}$ The possible successors are

$$t_0 \mathbin{@} \alpha \,|\, \beta\, \theta \xrightarrow{\mathrm{c}} t_0 \mathbin{@} \alpha\, \theta \qquad t_0 \mathbin{@} \alpha \,|\, \beta\, \theta \xrightarrow{\mathrm{c}} t_0 \mathbin{@} \beta\, \theta$$

  and by Proposition 18, $\mathrm{dsem}^{(n)}(t_0 \mathbin{@} \alpha \,|\, \beta\, \theta) = \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \alpha\, \theta) \cup \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \beta\, \theta)$.

  (2). Suppose without loss of generality that $t \in \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \alpha\, \theta)$. By $\mathbf{p}(n, t_0 \mathbin{@} \alpha\, \theta)$, $t_0 \mathbin{@} \alpha\, \theta \to^*_{s,c} t \mathbin{@} \varepsilon$. Prefixing this execution with the initial state by the disjunction rule, we obtain an execution from $t_0 \mathbin{@} \alpha \,|\, \beta\, s$ to $t \mathbin{@} \varepsilon$ as we wanted.

  (3). Any $t_0 \mathbin{@} \alpha \,|\, \beta\, \theta \to^*_{s,c} t \mathbin{@} \varepsilon$ must start with a $\to_c$ transition to $t_0 \mathbin{@} \alpha\, \theta$ or $t_0 \mathbin{@} \beta\, \theta$. Suppose we are in the second case, by induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} \beta\, \theta)$, $t \in \mathrm{dsem}^{(n)}(t_0 \mathbin{@} \beta\, \theta)$ but this is included in the $\mathrm{dsem}^{(n)}$ set of the initial term.

  (4). If $\bot$ is in the $\mathrm{dsem}^{(n)}$ of the initial set, it must be in the semantic set of one of its descendants, so there is an execution with call depth at least n from it. Prepending $t_0 \mathbin{@} \alpha \,|\, \beta\, \theta$ as initial state does not affect the call depth, so we have (4).

  (5). All derivations from the successor terms above being finite, the derivation from the disjunction only increases the length in one.

- $\boxed{\alpha^*}$ The possible successors are

$$t_0 \mathbin{@} \alpha^*\, \theta \xrightarrow{\mathrm{c}} t_0 \mathbin{@} \theta \xrightarrow{\mathrm{c}} t_0 \mathbin{@} \varepsilon \qquad t_0 \mathbin{@} \alpha^*\, \theta \xrightarrow{\mathrm{c}} t_0 \mathbin{@} \alpha\alpha^*\, \theta$$

and if $t \in \text{dsem}^{(n)}(t_0 \mathbin{@} \alpha^*\theta) = [\![\alpha^*]\!]^{(n)}(\theta, t_0)$ then there is a $k \in \mathbb{N}$ such that $t \in [\![\alpha]\!]^{(n)^k}(\theta, t_0)$. We can always use induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$.

(2). By the induction hypothesis and Lemma 22 we obtain (2).

(3). Take an execution $t_0 \mathbin{@} \alpha^*\,\theta \to^*_{s,c} t' \mathbin{@} \varepsilon$. An inductive proof will be sketched. There are only two possibilities for the second state of the derivation: (A) $t_0 \mathbin{@} \theta$, in which case the only reachable t is t_0 and it belongs to $[\![\alpha^*]\!]^{(n)}(\theta, t_0)$, and $t_0 \mathbin{@} \alpha\alpha^*\,\theta$. In this case, and using Lemma 21 (item 3), there must be a term t_m such that $t_0 \mathbin{@} \alpha\,\theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and (B) $t_m \mathbin{@} \alpha^*\theta \to^*_{s,c} t \mathbin{@} \varepsilon$. Both executions' call depth is at most n, so the induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$ lets us conclude that $t_m \in [\![\alpha]\!]^{(n)}(\theta, t_0)$ and, since $t_m \mathbin{@} \alpha^*\theta \to^*_{s,c} t' \mathbin{@} \varepsilon$ is strictly shorter than the original execution, we can assert $t \in [\![\alpha^*]\!]^{(n)}(\theta, t_m)$. Then joining these two memberships and by the semantic definition of α^*, $t \in [\![\alpha^*]\!]^{(n)}(\theta, t_0)$.

(4). If $\bot \in \text{dsem}^{(n)}(t_0 \mathbin{@} \alpha^*\,\theta)$, either there is an infinite iteration from $t_0 \mathbin{@} \alpha^*\,\theta$ or there is a k_0 such that $\bot \in [\![\alpha]\!]^{(n)^{k_0}}$. In the first case, we are done. Otherwise, by Lemma 24 there is a $k_\bot \in \mathbb{N}$ and a term $t_\bot \in [\![\alpha]\!]^{(n),k_\bot}(\theta, t)$ such that $\bot \in [\![\alpha]\!]^{(n)}(\theta, t_\bot)$. By induction hypothesis and Lemma 22, $t_0 \mathbin{@} \alpha^*\,\theta \to^*_{s,c} t_\bot \mathbin{@} \varepsilon$ and this must end with $t_\bot \mathbin{@} \alpha^*\,\theta \to_c t_\bot \mathbin{@} \theta \to_c t_\bot \mathbin{@} \varepsilon$. Using induction hypothesis $\mathbf{p}(n, t_\bot \mathbin{@} \alpha\,\theta)$, there is an execution whose call depth is at least $n+1$ from $t_\bot \mathbin{@} \alpha\,\theta$. Assembling both using Lemma 21 (item 1) and other rules and extensions, we conclude with

$$t_0 \mathbin{@} \alpha^*\,\theta \to^*_{s,c} t_\bot \mathbin{@} \alpha^*\,\theta \to_c t_\bot \mathbin{@} \alpha\alpha^*\,\theta \to^*_{s,c} \ldots$$

whose call depth is at least $n+1$.

(5). If $\bot \notin \text{dsem}^{(n)}(t_0 \mathbin{@} \alpha^*\,\theta)$ we are sure there is an k_0 such that $[\![\alpha]\!]^{(n),k_0}(\theta, t) = \emptyset$. This implies there are not execution sequences which unrolls α more than n_0 times, and so the maximum length of any execution is at most n_0 times the maximum length of an execution from $t \mathbin{@} \alpha\,\theta$, a finite number. The proof of the fact that α is not unrolled more that n_0 times is given by Lemma 23 (its assumption is true by induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$), because to unroll a $n_0 + 1$ time there must be a state $t \mathbin{@} \alpha^*\theta$ with $t \in [\![\alpha]\!]^{(n),k_0}(\theta, t) = \emptyset$ and this is impossible.

- $\boxed{\alpha \mathbin{?} \beta : \gamma}$ The possible successors are these two, but the second line can only be taken if the execution tree from $t_0 \mathbin{@} \alpha\,\theta$ is finite and does not contain solutions

$$t_0 \mathbin{@} \alpha \mathbin{?} \beta : \gamma\ \theta \xrightarrow{c} t_0 \mathbin{@} \alpha\beta\,\theta$$
$$\xrightarrow{c} t_0 \mathbin{@} \gamma\,\theta$$

And $\text{dsem}^{(n)}(t_0 \mathbin{@} \alpha \mathbin{?} \beta : \gamma\ \theta) = [\![\alpha \mathbin{?} \beta : \gamma]\!]^{(n)}(\theta, t_0)$ is $[\![\alpha]\!]^{(n)} \circ [\![\beta]\!]^{(n)}(\theta, t_0)$ if $[\![\alpha]\!]^{(n)}(\theta, t_0) \neq \emptyset$ and $[\![\gamma]\!]^{(n)}(\theta, t_0)$ otherwise.

(2). In the first case, since $t \in [\![\alpha]\!]^{(n)} \circ [\![\beta]\!]^{(n)}(\theta, t_0)$, there is a term t_m such that $t_m \in [\![\alpha]\!]^{(n)}(\theta, t_0)$ and $t \in [\![\beta]\!]^{(n)}(\theta, t_m)$. By induction hypothesis $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$ and $\mathbf{p}(n, t_m \mathbin{@} \beta\,\theta)$ there are executions $t_0 \mathbin{@} \alpha\,\theta \to^*_{s,c} t_m \mathbin{@} \varepsilon$ and $t_m \mathbin{@} \beta\,\theta \to^*_{s,c} t \mathbin{@} \varepsilon$. By Lemma 21, $t_0 \mathbin{@} \alpha\beta\,\theta \to^*_{s,c} t_m \mathbin{@} \beta\,\theta \to^*_{s,c} t \mathbin{@} \varepsilon$ as we wanted.

In the second case, by $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$ and $\mathbf{p}(n, t_0 \mathbin{@} \gamma\,\theta)$, $t_0 \mathbin{@} \gamma\,\theta \to^*_{s,c} t \mathbin{@} \varepsilon$, to which we append the [else] rule application because the execution tree from α is finite without solutions. Indeed, $\text{dsem}^{(n)}(t_0 \mathbin{@} \alpha\,\theta) = [\![\alpha]\!](\theta, t_0) = \emptyset$ and by property (5) all execution from $t_0 \mathbin{@} \alpha\,\theta$ are finite with call depth at most n, and by property (3) and the fact that $\bot \notin \emptyset$, no execution from α leads to a solution.

(3). Take a derivation $t_0 \mathbin{@} \alpha \mathbin{?} \beta : \gamma\ \theta \to^*_{s,c} t \mathbin{@} \varepsilon$. The second state can be $t_0 \mathbin{@} \alpha\beta\,\theta$ or $t_0 \mathbin{@} \gamma\,\theta$, in the latter case with a finite execution tree without solutions from $t \mathbin{@} \alpha\,\theta$. Anyhow we can apply induction hypothesis to the sequent (less strategy constructors). In the first case, t is in $\text{dsem}^{(n)}(t_0 \mathbin{@} \alpha\beta\,\theta)$ and this set is contained in the $\text{dsem}^{(n)}$ value of the original state, so t is in it.

In the second case, we know by (3) in $\mathbf{p}(n, t_0 \mathbin{@} \gamma\,\theta)$ that $t \in [\![\gamma]\!]^{(n)}(\theta, t_0)$. We want to prove that $R \coloneqq [\![\alpha]\!]^{(n)}(\theta, t_0)$ is empty, to apply the semantic definition for the conditional and finally conclude that t belongs to its semantic set. If $\bot \notin R$, by property (5) of $\mathbf{p}(n, t_0 \mathbin{@} \alpha\,\theta)$, all derivations from α have call depth at most n, and by property (3), no term belongs to R. Without terms and

without $\bot$, $R = \emptyset$. Otherwise $\bot \in R$, and there is an execution from α whose call depth is at least $n+1$. So the execution tree makes the initial execution call depth greater that n, and there is nothing to prove.

(4). If $\bot$ is in the global $\text{dsem}^{(n)}$, $\bot \in [\![\alpha]\!]^{(n)}$, or $[\![\alpha]\!]^{(n)} \neq \emptyset$ and $\bot \in [\![\beta]\!]^{(n)}$, or $[\![\alpha]\!]^{(n)} = \emptyset$ and $\bot \in [\![\gamma]\!]^{(n)}$. In any case and by the induction hypothesis we have invoked before, an execution whose call depth is greater than n can be built, perhaps using Lemma 21.

(5). If $\bot$ is not a member of the global dsem, all executions from α are finite and their call depth is at most n, and if $[\![\alpha]\!]^{(n)} \neq \emptyset$ then all executions from β are finite, and if $[\![\alpha]\!]^{(n)} = \emptyset$ then all executions from γ are finite. The executions from $t_0 \mathbin{@} \alpha \ \texttt{?}\ \beta : \gamma \ \theta$ are only prepended by this initial state, so they are also finite. The same arguments are valid for the call depth.

- $\boxed{\texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \ldots, x_n\ \texttt{using}\ \alpha_n}$ First, all possible reductions are

$$\begin{aligned} t_0 \mathbin{@} \texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \ldots, x_n\ \texttt{using}\ \alpha_n\, \theta \\ \to_c \text{subterm}(x_1 : \sigma(x_1) \mathbin{@} \alpha_1\sigma, \ldots, x_n : \sigma(x_n) \mathbin{@} \alpha_n\sigma; t) \mathbin{@} \theta \end{aligned}$$

for any $\sigma \in \mathsf{match}(P, t, C)$. The right-hand state is smaller that the left-hand state (a strategy constructor less), so we can apply the induction hypothesis. From the other side, the semantics of the first state equals the semantics of the second by the definition of the dsem, since the semantics of the substates $\text{dsem}^{(n)}(q_i) = \text{dsem}^{(n)}(\sigma(x_i) \mathbin{@} \alpha_i\sigma) = [\![\alpha_i]\!]^{(n)}(\sigma, \sigma(x_i))$ and both let expressions coincide. Their call depths are also the same. Thus, (2), (3), (4), and (5) intermediately follow.

- $\boxed{sl(t_1, \ldots, t_n)\, \theta)}$ The only possible successors of $t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta$ are $t_0 \mathbin{@} \delta_i\, \sigma\, \theta$ for any δ_i and $\sigma \in \mathsf{vmatch}(\vec{p}_i, \vec{s}, C)$ where $\vec{s} = \theta(t_1) \cdots \theta(t_n)$. From the other side

$$\begin{aligned} \text{dsem}^{(n)}(t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta) &= [\![sl(t_1, \ldots, t_n)]\!]^{(n)}(\theta, t_0) = f_{sl}^{(n)}(\theta(t_1), \ldots, \theta(t_n), t) \\ &= \bigcup_{(sl, \vec{p}_i, \delta_i, C_i) \in D} \bigcup \left\{ d_i^{(n)}(\sigma, t) : \sigma \in \text{vcheck}(\vec{p}_i, \vec{s}, C_i) \right\} \end{aligned}$$

Hence, $t \in \text{dsem}^{(n)}(t_0 \mathbin{@} sl(t_1, \ldots, t_m)\, \theta)$ if $t \in d_i^{(n)}(\sigma, t_0)$ for some $1 \leq i \leq |D|$.

(2). By the property (1) we have inductively (but separately) proved in the first paragraphs, and since $t \in d_i^{(n)}(\sigma, t_0)$, there must be an execution $t_0 \mathbin{@} \delta_i\, \sigma \to_{s,c}^* t \mathbin{@} \varepsilon$. θ can be added in the stacks' bottom, and using the call rule, we prove (2)

$$t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta \to_c t_0 \mathbin{@} \delta_i\, \sigma\, \theta \to_{s,c}^* t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$$

The rest of the properties are not proven uniformly in n and we will distinguish cases. First suppose $n = 0$. Then $d_k^{(0)} = \{\bot\}$. So $\text{dsem}^{(0)}(t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta) = \{\bot\}$ except in case $m_{sl} = 0$ or no definition matches the call arguments, when it is $\emptyset$.

(3). It holds vacuously because the executions from $t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta$ have call depth at least 1 and only execution whose call depth is 0 are considered.

(4). The reduction $t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta \to_c t_0 \mathbin{@} \delta_i\, \sigma\, \theta$ is allowed and its call depth is $1 > 0$.

(5). $\bot$ is in $\text{dsem}^{(0)}$, so there is nothing to prove.

Now, we will prove the case $n > 0$. By definition $d_i^{(n+1)} = F_i(d_1^{(n)}, \ldots, d_{|D|}^{(n)}) = [\![\delta_i]\!](d_1^{(n)}, \ldots, d_{|D|}^{(n)})$ so that

$$d_i^{(n+1)}(\sigma, t_0) = [\![\delta_i]\!]^{(n)}(\sigma, t_0) = \text{dsem}^{(n)}(t_0 \mathbin{@} \delta_i\sigma)$$

We invoke $\mathbf{p}(n, t_0 \mathbin{@} \delta_i\, \sigma)$ and proceed

(3). Take an execution $t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta \to_{s,c}^* t \mathbin{@} \varepsilon$ whose call depth is at most $n+1$. The second state must be $t_0 \mathbin{@} \delta_i\, \sigma\, \theta$ for some i and θ, and its tail must be $t \mathbin{@} \sigma\, \theta \to_c t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$. The maximum $\text{cdepth}(q_m) - \text{cdepth}(q_n)$ (with $m \geq n$) in the definition of call depth is reached only with $n = 1$ because the call depth is 1 in the initial state, while it is at least two in the rest of the execution but the two last states, where it only decreases. So the execution $t_0 \mathbin{@} \delta_i\, \sigma \to_{s,c}^* t \mathbin{@} \varepsilon$, removing the last states and θ from the bottom, has a call depth of n. Hence, the induction hypothesis can be applied to conclude $t \in \text{dsem}^{(n)}(t_0 \mathbin{@} sl(t_1, \ldots, t_n)\, \theta)$.

(4). If $\bot \in [\![sl(t_1, \dots, t_n)]\!]^{(n+1)}(\theta, t_0)$ then by the definition recalled above $\bot \in d_i^{(n+1)}(\sigma, t_0)$ for some i and σ. So $\bot \in \text{dsem}^{(n)}(t_0 \mathbin{@} \delta_i \sigma)$ and there is an execution $t_0 \mathbin{@} \delta_i \sigma \to_{s,c}^* q \mathbin{@} s$ with call depth at least $n+1$, reached as the difference between the call depths of the state number u and the state number l with $u \geq l$. Moreover, the minimum call depth of every state is 1 except if $s = \varepsilon$, when the penultimate state is $q \mathbin{@} \sigma$. Hence, the call depth for l is at least 1 because if l were the last state, $u = l$ and the call depth would be 0 that cannot be $n+1$ for any $n \in \mathbb{N}$.

Let's build an execution with call depth at least $n+2$. Adding θ to the bottoms, we increase the call depth of all states in 1, then we prepend the initial state and get $t_0 \mathbin{@} sl(t_1, \dots, t_n)\,\theta \to_c t_0 \mathbin{@} \delta_i \sigma \theta \to_{s,c}^* q \mathbin{@} s\,\theta$. Clearly the maximum in the call depth definition is reached subtracting the call depth of the first state, because one is a minimum and the maximum for $\text{cdepth}(q_m)$ is taken in the bigger set possible. The call depth of the modified state u (now in position $u+1$) has increased in one, so, subtracting 1, the call depth of the new execution is at least $n+2$, unless the subtracting call depth for the original substitution was 0. But this is impossible as we have reasoned in the paragraph above.

(5). If $\bot \notin [\![sl(t_1, \dots, t_n)]\!]^{(n+1)}(\theta, t_0)$, then $\bot \notin \text{dsem}^{(n)}(t_0 \mathbin{@} \delta_i \sigma)$ for all i and match σ, so from these states all executions are finite (if any). The execution sequences starting from $t_0 \mathbin{@} sl(t_1, \dots, t_n)\,\theta$ are the previous ones prepended by $t_0 \mathbin{@} sl(t_1, \dots, t_n)\,\theta \to_c$, and adding θ at the bottom of their stacks. They can also be extended by $t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$ to drop the new context at the end. Anyhow, they are still finite. This process also increases their call depth in 1, so that they are at most $n+1$, which is valid. This can be justified as in (4).

- $\boxed{\text{subterm}(\dots, x_i : q_i, \dots; t_0) \mathbin{@} \varepsilon}$ and call this state q in the following. We know

$$\text{dsem}^{(n)}(q) = \mathsf{let}\ t_1 \leftarrowtail \text{dsem}^{(n)}(q_1) \dots \mathsf{let}\ t_k \leftarrowtail \text{dsem}^{(n)}(q_k) : \text{dsem}^{(n)}(t_0[x_i/t_i]_{i=1}^k \mathbin{@} \varepsilon)$$

So if $t \in \text{dsem}^{(n)}(q)$ there must be $t_i \in \text{dsem}^{(n)}(q_i)$ for all $1 \leq i \leq k$ such that $t = t_0[x_i/t_i]_{i=1}^k$.

(2). By induction hypotheses $\mathbf{p}(n, q_i)$ for every substate q_i of q, $q_i \to_{s,c}^* t_i \mathbin{@} \varepsilon$. Lemma 25 lets us conclude (2) because there is an execution $q \to_{s,c}^* t_0[x_i/t_i]_{i=1}^k \mathbin{@} \varepsilon = t \mathbin{@} \varepsilon$.

(3). Suppose there is an execution $q \to_{s,c}^* t \mathbin{@} \varepsilon$. By Lemma 25 we know $q_i \to_{s,c}^* t_i \mathbin{@} \varepsilon$ for some terms t_i and $t = t_0[x_i/t_i]_{i=1}^k$. Then by induction hypotheses $\mathbf{p}(n, q_i)$, $t_i \in \text{dsem}^{(n)}(q_i)$. Looking at the $\text{dsem}^{(n)}$ definition above and provided $\text{dsem}^{(n)}(t \mathbin{@} \varepsilon) = \{t\}$ we conclude $t \in \text{dsem}^{(n)}(q)$.

(4). If $\bot \in \text{dsem}^{(n)}(q)$ then $\bot \in \text{dsem}^{(n)}(q_l)$ for some $1 \leq l \leq k$. Then by hypothesis there is an execution $q_l \to_{s,c}^* q'$ whose call depth is at least $n+1$. We can then build a derivation from q advancing the l-th component by the reductions of the aforementioned execution using $[\text{prl}_s]$ or $[\text{prl}_c]$ (Lemma 25, item 1). Then a derivation from q whose call depth is at least $n+1$ exists.

(5). If $\bot \notin \text{dsem}^{(n)}(q)$, $\bot$ cannot be in any of the $\text{dsem}^{(n)}(q_i)$ by definition of let. Hence, all derivations from q_i are finite and so are those from q according to Lemma 25. The condition about the call depths is also preserved.

- $\boxed{\text{rewc}(p : q; \dots; t_0)}$ The semantics of this state is

$$\begin{aligned} R &:= \text{dsem}^{(n)}(\text{rewc}(p : q; \dots; t_0) \mathbin{@} \theta) \\ &= \mathsf{let}\ t' \leftarrowtail \text{dsem}^{(n)}(q) : \bigcup_{\sigma_m \in \mathsf{match}(p, t')} \mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}^{(n)}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta) : \{m(\sigma'(r))\} \end{aligned}$$

If t belongs to the set above, there must be a $t_m \in \text{dsem}^{(n)}(q)$, σ_m, and σ'. From the other side, the execution from the term can evolve using $[\text{rewc}_s]$ and $[\text{rewc}_c]$

$$\text{rewc}(p : q, \dots) \to_{s,c}^* \text{rewc}(p : q', \dots) \iff q \to_{s,c}^* q'$$

and if q' is a solution $t_m \mathbin{@} \varepsilon$, the following state must be one of two:

– C is equational and there is a match σ' as we can read in the last rule in Figure 3. The following state is $t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$ for some t.

– If C contains a rewriting fragment, the next state is another $\text{rewc}(rc : \sigma_1(lc) \mathbin{@} \alpha\,\theta, \sigma_1, C' \ldots) \mathbin{@} \theta$ where $C = C_0 \wedge lc \Rightarrow rc \wedge C'$ with C_0 an equational condition and σ_1 like in the last rule in Figure 2. This state is lower in the order because it has fewer condition fragments, so we can call $\mathbf{p}(n, \text{rewc}(rc : \sigma_1(lc) \mathbin{@} \alpha\theta, \sigma_1, C', \ldots))$.
More precisely, $\sigma_1 \in \mathsf{check}(C_0, \sigma \circ \sigma_m)$, and according to Proposition 18 (3) the semantics of this new rewriting state is $\mathsf{let}\ \sigma' \leftarrowtail \mathsf{check}(lc \Rightarrow rc \wedge C', \sigma_1; \bar{\alpha}, \theta) : \{m(\sigma'(r))\}$. By Lemma 26, $\sigma' \in \mathsf{check}^{(n)}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta)$. So all reachable states from the new rewriting state are in the semantics of the initial rewc state.

(2). Since $t \in \text{dsem}^{(n)}(q)$ and by induction hypothesis $\mathbf{p}(n, q)$, $q \to^*_{s,c} t_m \mathbin{@} \varepsilon$ so

$$\text{rewc}(p : q, \sigma, C, \bar{\alpha}, \ldots) \mathbin{@} \theta \to^*_{s,c} \text{rewc}(p : t_m \mathbin{@} \varepsilon, \sigma, C, \bar{\alpha}, \ldots) \mathbin{@} \theta$$

and there are $\sigma_m \in \mathsf{match}(p, t_m)$ and $\sigma' \in \mathsf{check}^{(n)}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta)$ such that $t = m(\sigma'(r))$. Using the previous and if C is equational, we can apply the rule we have mentioned above and expand this derivation with $\to_s t \mathbin{@} \theta \to_c t \mathbin{@} \varepsilon$ as we wanted. Otherwise, if $C = C_0 \wedge lc \Rightarrow rc \wedge C'$ and $\sigma' \in \mathsf{check}^{(n)}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta)$ there must be a $\sigma_1 \in \mathsf{check}^{(n)}(C_0, \sigma \circ \sigma_m)$ by Lemma 26 such that $\sigma' \in \mathsf{check}^{(n)}(C', \sigma_1, \bar{\alpha}, \theta)$. So we can apply the rule cited above and append at the end of the execution $\text{rewc}(rc : \sigma_1(lc) \mathbin{@} \alpha\theta, \sigma_1, C', \ldots)$. Applying the induction hypothesis to this state, we get the rest of the derivation and conclude.

(3). Take a derivation $\text{rewc}(p : q, \ldots) \to_c t \mathbin{@} \varepsilon$. There must be a first state $\text{rewc}(p : t_m \mathbin{@} \varepsilon, \ldots)$ and $q \to^*_{s,c} t_m \mathbin{@} \varepsilon$. Hence and by induction hypothesis, $t_m \in \text{dsem}^{(n)}(q)$. If the next state is a term state $t \mathbin{@} \theta$, the rule from Figure 3 must have been applied so there is a σ' to conclude that $t \in \text{dsem}^{(n)}(\text{rewc}(\ldots))$.
When the next state is another rewriting condition state, the rule in Figure 2 must have been applied. So there are σ_m and σ_1. According to what we have said in the initial paragraphs of this case, all terms in $\text{dsem}^{(n)}$ of the new state are in $\text{dsem}^{(n)}$ of the initial state. And by induction hypothesis, the reachable solutions from the state are in its $\text{dsem}^{(n)}$.

(4). If $\bot \in R$ then $\bot \in \text{dsem}^{(n)}(q)$ or $\bot \in \mathsf{check}^{(n)}(C, \sigma \circ \sigma_m; \bar{\alpha}, \theta)$ (in which case there must be a rewriting condition fragment in C). If the first is true, there is an execution from q whose call depth is at least $n + 1$. Using $[\text{rewc}_s]$ and $[\text{rewc}_c]$, we can easily build an execution from the rewc state with the same call depth.
In the second case $\bot$ is in the $\text{dsem}^{(n)}$ of the next condition fragment state, so by induction hypothesis there is an execution from it whose call depth is at least $n + 1$, and prefixing the way to the lower state preserves the call depth. So we have an execution from the initial state with a call depth greater than n.

(5). If $\bot \notin R$, all executions from q are finite with call depth at most n. If C does not contain rewriting fragments, the executions' length can only be extended by two new states in decreasing call depths. Otherwise, all derivations get stuck or arrive to the next rewriting fragment state. By induction hypothesis, all executions from it are finite and with call depth bounded by n, so the full execution also satisfies these properties.

■

▸ **Corollary 28.** For all $t, t' \in T_\Sigma(X)$, $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $\theta \in \text{VEnv}$

$$t' \in [\![\alpha]\!](\theta, t) \quad \iff \quad t \mathbin{@} \alpha\,\theta \to^*_{s,c} t' \mathbin{@} \varepsilon$$

and $\bot \in [\![\alpha]\!](\theta, t)$ iff there is an infinite derivation from $t \mathbin{@} \alpha\,\theta$.

It follows immediately from the previous theorem, by taking $q := t_0 \mathbin{@} \alpha\,\theta$.

■

A.3.1 Some properties

▸ **Definition.** All recursive calls in a strategy expression are tail if the expression is:

- `idle`, `fail`, a test, or a strategy call expression.

- $\alpha \,|\, \beta$ and all recursive calls in α and β are tail.
- $\alpha \,;\, \beta$ and α does not contain recursive calls and all recursive calls in β are tail.
- $\alpha \,?\, \beta : \gamma$ and α does not contain recursive calls, and all recursive calls in β and γ are tail.
- A subterm rewriting or rule application expression, and all recursive calls in its substrategies are tail.

▸ **Proposition 29.** For any $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$. The set of reachable states from $t \,@\, \alpha$ is finite for all $t \in T_\Sigma(X)$ if any of the following conditions holds:

1. α does not contain iterations or recursive function calls.
2. The reachable terms from $t \,@\, \alpha$ are finitely many, strategy call argument only take a finite number of values, and all recursive calls in strategy definitions are tail.

   We must also consider that variable environments are replaced instead of pushed to the stack when other substitution is on top. This *optimization* does not affect the semantics.

In the first statement, no recursive strategy calls are allowed, but non-recursive ones are admissible. This means that the call graph of the strategy definitions does not contain cycles: it is a directed acyclic graph. Thus, it admits a topological ordering, in which the expression α can be included. We proceed by induction on this topological ordering and then on the structure of the execution states using the using the lexicographic tuple in the proof of the last theorem (without approximants):

- For $t \,@\, \texttt{idle}\, \theta$, $t \,@\, \texttt{fail}\, \theta$, $t \,@\, \texttt{match}\ P\ \texttt{s.t}\ C\, \theta$ only $t \,@\, \theta$, $t \,@\, \varepsilon$, and themselves can be reached by $\to_{s,c}$. So the reachable states' cardinality is at most two.
- $t \,@\, \alpha \,|\, \beta\, \theta$ can only be followed by $t \,@\, \alpha\, \theta$ and $t \,@\, \beta\, \theta$. By induction hypothesis the reachable states from both are finite and so is their union.
- $t \,@\, \theta\, s$ is only followed by $t \,@\, s$. By induction hypothesis the reachable states are finitely many.
- $t \,@\, \alpha\, s$. We know by Lemma 21 that if $\theta = \text{vctx}(s)$

  $$\{q : t \,@\, \alpha\, s \to^*_{s,c} q\} = \{q \,@\, s' s : t \,@\, \alpha\, \theta \to^*_{s,c} q \,@\, s'\} \cup \{q : t \,@\, \alpha\, \theta \to_{s,c} t_m \,@\, \varepsilon \ \wedge\ t_m \,@\, s \to^*_{s,c} q\}$$

  By induction hypothesis for $t \,@\, \alpha\, \theta$, the number of q in the first set is finite. Thus, the number of t_m in the second set is finite too, and by induction hypothesis for $t_m \,@\, s$, a finite number of q lay in the second set. Hence, the global number of reachable states is finite.
- $t \,@\, \alpha \,;\, \beta\, \theta$. The only direct successor is $t \,@\, \alpha\beta\, \theta$, which is lower in the order. The case is already covered by induction hypothesis.
- $t \,@\, rl[Subs]\, \theta$ has a finite number of successors, as a finite number of rules (with label rl) can be applied in a finite number of positions, with a finite number of matches. If the rule application includes strategies for the rewriting condition fragments, it is still finite. $t \,@\, rl[Subs]\{\alpha_1, \dots, \alpha_n\}\theta \to_c \text{rewc}(p : q, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) \,@\, \theta$ is the only possibility for a finite set of σ, θ_s, r and m. The sequent is lower in the order, so by induction hypothesis, only a finite number of states are reachable from it.
- $t \,@\, \texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \dots, x_n\ \texttt{using}\ \alpha_n \to_c \text{subterm}(\dots, x_i : t \,@\, \alpha_i, \dots; t) \,@\, \theta$ for all possible matches, which are a finite set. Being the successor lower in the order and by induction hypothesis, the reachable states are finitely many.
- For $t \,@\, \alpha\ ?\ \beta\ :\ \gamma\ \theta$ and by induction hypothesis both $t \,@\, \alpha\, \theta$, $t' \,@\, \beta\, \theta$ and $t \,@\, \gamma\, \theta$ for any term t' reach only a finite number of states. The successors of $t \,@\, \alpha\ ?\ \beta\ :\ \gamma$ by $\to_c$ are $t \,@\, \gamma\, \theta$ and $t \,@\, \alpha\beta\, \theta$. Both of them have a finite number of successors by hypothesis and an already considered case.
- For $\text{subterm}(x_1 : q_1, \dots, x_n, q_n; t) \,@\, s$, induction hypothesis can be applied to q_i so the reachable states from them are finitely many. In Lemma 25 we learn that the reachable states from a subterm state are the combination of its reachable subterms (which is a finite number, the product of the number of reachable substates) plus the reachable states from any $\text{subterm}(x_1 : t_1 \,@\, \varepsilon, \dots, x_n : t_n \,@\, \varepsilon; t) \,@\, s \to_c t[x_i/t_i]_{i=1}^n \,@\, s$. By induction hypothesis for $t' \,@\, s$, these are a finite number. Thus, the reachable states from the original one are finitely many.

- For $\mathrm{rewc}(p : q, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) @ s$, we can invoke the induction hypothesis for q. Again the number of reachable states induced by the execution of q is finite. For every final execution $\mathrm{rewc}(p : t_m @ \varepsilon, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) @ s$ there are a finite number of direct successors, one for each possible match. These successors are rewc states with fewer condition fragments or plain states $t' @ s$. In any case we can apply induction hypothesis to conclude that the reachable states are finitely many.
- For a strategy call, $t @ sl(t_1, \dots, t_n)\theta \to_c t @ \delta_i \sigma\, \theta$ for every $\sigma \in \mathsf{vmatch}(\vec{p}_i, (t_1, \dots, t_n), C_i)$ and definition $(sl, \vec{p}_i, \delta_i, C_i) \in D$, which are finitely many. Since the definition δ_i is below in the topological order, we known that finitely many states are reachable when executing it, so the reachable states from the strategy call are a finite union of finite sets, hence finite.

The second statement will be proved differently. First, observe that the aforementioned optimization does not affect the semantics, because $\mathrm{vctx}(\sigma_1\sigma_2 s) = \sigma_1$ whatever σ_2 is and the only allowed execution from $t @ \sigma_1\sigma_2\, s$ starts with $t @ \sigma_1\sigma_2\, s \to_c t @ \sigma_2\, s \to_c t @ s$. Thus, σ_2 do not have any effect and can be safely removed. Notice that recursive calls in iterations α^* are never tail, so no recursive calls are allowed inside α^*. We can then prove regardless of recursive calls that starting from $t @ \alpha^*\theta$ there is a finite number of reachable states. This will be done by induction on the number of nested iterations in α. Suppose α does not contain another iteration, then by the first statement, $t' @ \alpha\, \theta$ reaches a finite number of states for any term t'. Consider the set R including every reachable states from $t' @ \alpha^*\theta$ for every reachable term t', which clearly contains the reachable states from $t @ \alpha^*\theta$. An element of this set is either $t' @ \alpha^*\theta$, $t' @ \theta$, $t' @ \varepsilon$, or a reachable state from $t' @ \alpha\, \theta$ with $\alpha^*\theta$ instead of θ at the bottom of the stack for some t'. Since the latter are finitely many and the number of t' is also finite, R is a finite set, and we are done. If α contains an iteration β^*, the number of nested iterations in β is strictly lower that the number for α, so the induction hypothesis lets us conclude that the reachable states from β^* are finitely many. A straightforward induction on the structure of the expressions, like in the first statement, would complete the proof.

If all recursive calls in δ_i are tail, we will prove that the reachable states are finitely many. When a strategy is called, $t @ sl(t_1, \dots, t_n)s \to_c t @ \delta_i \sigma s$, new content is pushed on top of the stack: all calls in δ_i are tail and σ only take a finite number of values since strategies are only called with finitely many distinct arguments by hypothesis and matches for each one are also finite. Reachable states from $t @ \delta_i\sigma s$ include the reachable states from the state itself without executing recursive calls, and the reachable states from $t' @ sl(t_1, \dots, t_n)\sigma s$ for any recursive call in δ_i and terms t' (remember all recursive calls are tail). The first set of reachable states is finite, because of the previous results and provided that recursive calls are not expanded, and the $t' @ sl(t_1, \dots, t_n)\sigma s$ states are also finitely many because, by assumption, reachable terms t' and call arguments $t_1, \dots, t_n$ take a finite number of values. These states evolve $t' @ sl(t_1, \dots, t_n)\sigma s \to_c t' @ \delta_i\sigma'\sigma s$ but with the optimization the second state is replaced by $t' @ \delta_i\sigma' s$, which was already counted. Hence, a finite number of states is reachable from all strategy definitions.

The proof of the previous statement of the proposition can be used to conclude that all reachable states from any original state $t @ \alpha$ are finitely many.

■

▸ **Proposition 30.** The ω-language $E(\alpha, I)$ where finite words are extended repeating the last state forever is

$$\{\mathrm{cterm}(q_0)\,\mathrm{cterm}(q_1) \cdots \mathrm{cterm}(q_n) \cdots \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots\}$$

where the $\twoheadrightarrow$ transition is extended with $q \twoheadrightarrow t @ \varepsilon$ whenever $q \to_c^* t @ \varepsilon$.

Let $E_\omega(\alpha, I)$ be the language $E(\alpha, I)$ where finite words are extended repeating the last symbol forever, and S the set in the statement. We have to prove that $E_\omega(\alpha, I) = S$. Take $w \in E_\omega(\alpha, I)$, two cases are possible for some $t \in I$:

- $w \in T_\omega(t @ \alpha)$. Here, S and the definition of T_ω coincide, so clearly $w \in S$.
- $w = w_1 \cdots w_n w_n^\omega$ where $w_1 \cdots w_n \in T_{\mathrm{succ}}(t @ \alpha)$. By definition of T_{succ}, there are $q_1, \dots, q_n$ such that $q_k \twoheadrightarrow q_{k+1}$, $w_k = \mathrm{cterm}(q_k)$, and $q_n \to_c^* w_n @ \varepsilon$.

  Then, defining $q_k = w_n @ \varepsilon$ for all $k \geq n + 1$, we have $q_n \twoheadrightarrow q_{n+1}$ and $q_k \twoheadrightarrow q_{k+1}$ in the extended sense for all $k \geq n + 1$, because $w_n @ \varepsilon \to_c^* w_n @ \varepsilon$ with an empty execution. Hence, $w \in S$.

From the other side, take $w \in S$. By its definition, there are q_k for $k \geq 1$ such that $q_k \twoheadrightarrow q_{k+1}$ and $w_k = \text{cterm}(q_k)$. Suppose all $\twoheadrightarrow$ reductions are of the original $\twoheadrightarrow$ relation, then the definition of T_ω is attained, so $w \in E_\omega(\alpha, I)$. Otherwise, there is an $l \geq 1$ such that $q_l \twoheadrightarrow q_{l+1}$ only in the extended relation. Then $q_l \to_c^* w_l \,@\, \varepsilon$ and $q_{l+1} = w_l \,@\, \varepsilon$. No control reduction can be applied to $w_l \,@\, \varepsilon$, so the only allowed execution from the state is the empty one. Thus, q_{l+2} must be q_{l+1} itself, and inductively $q_k = q_{l+1} = w_l \,@\, \varepsilon$ for all $k \geq l+1$. Finally, $w_k = w_l$ for all $k \geq l$ and $w_1 \cdots w_l$ is in $T_{\text{succ}}(t \,@\, \varepsilon)$ because $q_l \to_c^* w_l \,@\, \varepsilon$. All that means $w \in E_\omega(\alpha, I)$.

■

▸ **Proposition 31.** For any $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ finite, if the reachable states from $t \,@\, \alpha$ are finitely many for $t \in I$, the extended abstract strategy $E(\alpha, I)$ is a ω-regular language, and a Büchi automaton for it is $\mathcal{A} = (Q, \text{cterm}(Q), \delta, \{start\}, Q)$ where $Q = \{start\} \cup \{q \in \mathcal{X}S_\mathcal{R} \mid t \in I \ \wedge \ t \,@\, \alpha \twoheadrightarrow^* q\}$ and

$$\begin{aligned}
\delta(start, t) &= \{\ t \,@\, \alpha\ \} && \text{if } t \in I \\
\delta(start, t) &= \emptyset && \text{if } t \notin I \\
\delta(q, t) &= \{\ q' : q \twoheadrightarrow q' \ \wedge\ \text{cterm}(q') = t\ \} && \text{for } q \in Q \backslash \{start\} \\
&\quad \cup \{\ t \,@\, \varepsilon : \text{if } q \to_c^* t \,@\, \varepsilon\ \}
\end{aligned}$$

Then, we must check that $L(\mathcal{A}) = E(\alpha, I)$. First prove $L(\mathcal{A}) \subseteq E(\alpha, I)$. For each $w \in L(\mathcal{A})$ there must be an automaton run $start\,\pi \in \mathcal{X}S_\mathcal{R}^\omega$. Let ensure $\pi_i \twoheadrightarrow \pi_{i+1}$ and $w_i = \text{cterm}(\pi_i)$, so that $w \in E(\alpha, I)$ by definition.

- Base case: the automata starts in the state *start* so w_0 must belong to I and $\pi_0 = w_0 \,@\, \alpha$. Clearly $w_0 = \text{cterm}(w_0 \,@\, \alpha)$.
- Inductive case: for $i \geq 1$, $\pi_{i+1} \in \delta(\pi_i, w_{i+1})$ so by definition of δ, $\pi_i \twoheadrightarrow \pi_{i+1}$ and $w_{i+1} = \text{cterm}(\pi_{i+1})$ if π_{i+1} is in the first set. If it is in the second, $\pi_i \to_c^* \pi_{i+1} = w_{i+1} \,@\, \varepsilon$, and $w_{i+1} = \text{cterm}(\pi_i)$ because the control reductions preserve the current term. Hence, $w_{i+1} = \text{cterm}(\pi_{i+1}) = \text{cterm}(\pi_i) = w_i$ and $\pi_i \twoheadrightarrow \pi_{i+1}$ in the extended sense.

Then we need to prove $E(\alpha, I) \subseteq L(\mathcal{A})$. If $w \in E(\alpha, I)$ then $w \in T_{\text{succ}}(t \,@\, \alpha)$ or $w \in T_\omega(t \,@\, \alpha)$ for some $t \in I$. Let do the second case, the first is quite similar. By definition of T_ω there is a succession of states $q_0 \cdots q_n \cdots$ such that $w_i = \text{cterm}(q_i)$ with $q_0 = t \,@\, \alpha$. This succession (prepended by *start*) is a run for $\mathcal{A}$:

- Base case: $q_0 = w_0 \,@\, \alpha$ with $w_0 \in I$. So the automaton can pass from *start* to q_0.
- Inductive case: the automaton is in the state q_i and reads w_{i+1}. We know that $q_i \twoheadrightarrow q_{i+1}$ and $w_{i+1} = \text{cterm}(q_{i+1})$ so $q_{i+1} \in \delta(q_i, w_i)$.

And consequently $w \in L(\mathcal{A})$.

■

A.3.2 Another definition for strategies

▸ **Lemma 32.** If $t \,@\, s \to_{s,c}^+ t' \,@\, s'$, the number of strategy constructors is strictly lower in s' than in s, or the execution reduces a strategy call and $\text{cdepth}(t' \,@\, s') > \text{cdepth}(t \,@\, s)$, or s and s' contain an iteration expression that is being executed.

We only have to observe Figures 2 and 3 to see that all rules from flat states, except the rule for calls and starting iterations, reduce the number of strategy constructors. Some other rules create subterm or rewc states, but they also reduce the number of constructors in the strategy stack. The rules that modify these structured states preserve the stack unchanged, so when they are removed the stack stay with a strictly lower number of constructors.

The rule for strategy calls can increment the number of strategy constructors, because it pushes the strategy definition on top of the stack. However, it also pushes a substitution, thus increasing the call depth of the successor state. The substitution is the first item to be pushed, so it is not removed until all the strategy definition has been popped. And then, the number of strategy constructor shrinks, because the strategy call term was removed.

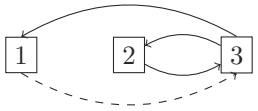
■

▸ **Proposition 33.** Given an execution $\pi = q_0 \cdots q_n \cdots \in \mathrm{Ex}_\omega(q_0)$, the following conditions are equivalent:

1. For all $n \in \mathbb{N}$ and position p in q_n such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, $\mathrm{Iter}(p, \alpha, s)$ is finite.
2. π iterates finitely.
3. For any $n \in \mathbb{N}$ and position p in q_n such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, $\mathrm{Iter}(p, \alpha, s)$ is finite or $\mathrm{Leave}(p, \alpha, s)$ is infinite.

And the last two conditions are equivalent even with the optimization of Proposition 4:2.

This is the plan for the proof. Note that the implication (1) ⇒ (3) is obvious, and that only (3) ⇒ (1) is not valid when the optimization of Proposition 4:2 is considered.



- (2) ⇒ (3). Take n and p such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$. By *reductio ad absurdum*, suppose $\mathrm{Leave}(p, \alpha, s)$ is finite and $\mathrm{Iter}(p, \alpha, s)$ is infinite. There exists $M := \max \mathrm{Leave}(p, \alpha, s)$, and since $\mathrm{Iter}(p, \alpha, s)$ is infinite, it must have an element $k > M$. Then, $\mathrm{ConsIter}(k, p) = \mathrm{Iter}(p, \alpha, s) \cap [k, \infty)$, which is an infinite set, against (2).
- (3) ⇒ (2). For any n and p such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, we need to prove that $\mathrm{ConsIter}(n, p)$ is finite. Since $\mathrm{ConsIter}(n, p) \subseteq \mathrm{Iter}(p, \alpha, s)$, if the second is finite, the first is.

  Otherwise $\mathrm{Leave}(p, \alpha, s)$ is infinite, so there is a $j \geq n$ such that $j \in \mathrm{Leave}(p, \alpha, s)$. Thus, $\mathrm{ConsIter}(n, p)$ size is less that $j - n + 1$, so it is finite.
- (3) ⇒ (1). For any n and p such that $q_n|_p \to_c^* t_n \mathbin{@} \alpha^* s$, there are two possibilities, either $\mathrm{Iter}(p, \alpha, s)$ is finite and we are done, or $\mathrm{Leave}(p, \alpha, s)$ is infinite. We will prove that the second option is not possible, but this is only valid when not using the optimization. First, we claim that states of the form $t \mathbin{@} s$ can only appear a finite number of times in an execution, from which the finiteness of Leave follows. This fact is not true when using the optimization from Proposition 4:2, because call contexts disappear and cannot be used to distinguish states.

  If we have $t \mathbin{@} s \twoheadrightarrow^+ t' \mathbin{@} s$, since the number of strategy constructors and call depth in both sides is the same, and by Lemma 32, then s must contain an iteration expression that has been reduced during the execution. We proceed by induction on the number of iteration expressions in s. Suppose there is none, then $t \mathbin{@} s$ occur only once in the execution, so $\mathrm{Leave}(p, \alpha, s)$ is finite for all p and α involved.

  Otherwise, suppose $s = s_1 \beta^* s_0$ where s_1 does not contain iteration expressions, and the occurrences of $t \mathbin{@} s_0$ for some different terms t are finite. Then $\mathrm{Leave}(p, \beta, s_0)$ is finite, and by (3), $\mathrm{Iter}(p, \beta, s_0)$ also is. If we have $t \mathbin{@} s \twoheadrightarrow^+ t' \mathbin{@} s$, there must be an iteration of β in between, but only a finite number of these are possible, so only a finite number of states of the from $t \mathbin{@} s$ occur. Hence, $\mathrm{Leave}(p, \alpha, s)$ and $\mathrm{Iter}(p, \alpha, s)$ are finite, by statement (3).

■

▸ **Proposition.** Given an execution $\pi \in \mathrm{Ex}_\omega(q_0)$ that iterates finitely, $\mathrm{cdepth}(\pi_n) \to \infty$ when $n \to \infty$.

First, we prove $\mathrm{cdepth}(\pi_n) \to \infty$ if the number of strategy calls is infinite, provided π iterates finitely and the optimization is not used, as otherwise cdepth would not make sense. A strategy stack can be seen as a call stack. In $s_1 \theta s_0$, where s_1 does not contain substitutions, s_1 is the remaining *code* for the current call context, and θ are their local variables; s_0 is the rest of the backtrace, which may contain several other nested contexts. When a strategy call is reduced, another call context is created.

We will build a graph whose vertices are the different call contexts that occur in the execution of π. The edges link a context with its single parent context. This forms a finitary tree, as only a finite number of calls can appear in a context, i.e. in a strategy definition or in the main strategy, and they can only be executed a finite number of times in each context, because π iterates finitely.

If there are infinitely many strategy calls, the tree must be infinite. By Kőnig lemma, there must be an infinite branch. Remember that $\mathrm{cdepth}(\pi_n) \to \infty$ means that for all $M \in \mathbb{N}$ there is an $n_0 \in \mathbb{N}$ such that for

all $n \geq n_0$, $\text{cdepth}(\pi_n) \geq M$. Choosing the vertex at depth M in the infinite branch, we get a context from which the execution will never exit, as there is a non-terminating calling sequence in it.

Now, we want to see that π contains infinite strategy calls. By Lemma 32 and because reductions do not leave from states with empty stack, there must be an infinite number of iterations or an infinite number of strategy calls. If there is a finite number of strategy calls, there is a finite number of stacks. And only a finite number of iteration expressions can be found in π, since they are also finite in the strategy definitions. Hence, the states where an iteration starts are inside the union of all $\text{Leave}(p, \alpha, s)$ for p (depend on the number of `matchrew` in the terms), α and s. The set $\text{Leave}(p, \alpha, s)$ is finite by the first statement of Proposition 4:4, and their parameters can only take a finite number of values, as we have reasoned above. So, the number of iterations is finite. Thus, assuming that the number of calls is infinite leads to a contradiction, so the number of calls is infinite.

■

▸ **Proposition 34.** For any $\alpha \in \text{Strat}_{\mathcal{R},\Omega}$ and $I \subseteq T_\Sigma(X)$ finite, if the reachable states from $t \,@\, \alpha$ are finitely many for $t \in I$, the extended abstract strategy $E(\alpha, I)$ is an ω-regular language, and a Streett automaton for it is $\mathcal{A} = (Q^2, \text{cterm}(Q \setminus \{start\}), \delta, \{start\} \times I, S)$ where

$$Q = \{start\} \cup \{q \in \mathcal{X}S_{\mathcal{R}} \mid t \in I \ \wedge \ t \,@\, \alpha \twoheadrightarrow^* q\},$$

the Streett conditions are $S = \{\ (A_{p,\alpha,s}, B_{p,\alpha,s}) : (p, \alpha, s) \in J\ \}$ for

$$\begin{aligned} J &= \{\ (p, \alpha, s) : \exists\, q \in Q, t \in T_{\Sigma/E} \quad q|_p \to_c^* t \,@\, \alpha^* s\ \} \\ A_{p,\alpha,s} &= \{\ (q, q') \in Q^2 : q|_p \to_{s,c}^* t \,@\, \alpha\alpha^* s \twoheadrightarrow q'|_p,\ t \in T_{\Sigma/E}\ \} \\ B_{p,\alpha,s} &= \{\ (q, q') \in Q^2 : q|_p \to_{s,c}^* t \,@\, s \twoheadrightarrow q'|_p,\ t \in T_{\Sigma/E}\ \}, \end{aligned}$$

and

$$\delta((q, q'), t) = \{\ (q', q'') \in Q^2 : q' \twoheadrightarrow q'' \ \wedge \ \text{cterm}(q') = t\ \} \cup \{\ (q', t \,@\, \varepsilon) : \text{if } q' \to_c^* t \,@\, \varepsilon\ \}$$

This automaton is valid both with or without the optimization of Proposition 4:2.

Looking at the automaton in Proposition 4:3, we can see that $\{q : (y, q) \in \delta((x, y), t)\} = \delta_0(y, t)$ for any $x, y \in Q$ if δ_0 is the transition relation of the previous automaton. Hence, ignoring the additional copy of the next execution state in each state in this automaton, their runs are the same and they recognize the same words except for the acceptance condition. Thus, we only have to ensure that the additional restrictions actually ban executing iterations forever.

Let π be an execution of the semantics, we want to see that $\rho = (start, \pi_0)(\pi_0, \pi_1)\cdots$ is accepting iff π iterates finitely. Remember that ρ is accepting iff $\inf(\rho) \cap A_{p,\alpha,s} = \emptyset$ or $\inf(\rho) \cap B_{p,\alpha,s} = \emptyset$ for all $(p, \alpha, s) \in J$. And π iterates finitely if $\text{Iter}_\pi(p, \alpha, s)$ is finite or $\text{Leave}_\pi(p, \alpha, s)$ is infinite, for all $(p, \alpha, \beta) \in J$, by the third statement of Proposition 4:4. Comparing definitions, it is easy to conclude

$$\{\ (\pi_k, \pi_{k+1}) : k \in \text{Iter}_\pi(p, \alpha, s)\ \} = A_{p,\alpha,s} \cap P \qquad \{\ (\pi_k, \pi_{k+1} : k \in \text{Leave}(p, \alpha, s)\ \} = B_{p,\alpha,s} \cap P,$$

where $P = \{(\pi_k, \pi_{k+1}) : k \in \mathbb{N}\} = \{\rho_k : k \geq 1\}$. Now, intersecting with $\inf(\rho) \subseteq P$ in both equations and both sides, ρ is accepting if

$$\inf(\rho) \cap \{\ \rho_k : k \in \text{Iter}_\pi(p, \alpha, s)\ \} = \emptyset \quad \text{or} \quad \inf(\rho) \cap \{\ \rho_k : k \in \text{Leave}_\pi(p, \alpha, s)\ \} \neq \emptyset$$

Since $\inf(\rho)$ are the states q such that the number of $k \in \mathbb{N}$ satisfying $\pi_k = q$ is infinite, the latter equations mean $\text{Iter}_\pi(p, \alpha, s)$ is finite or $\text{Leave}_\pi(p, \alpha, s)$ is infinite. Hence, π iterates finitely iff ρ is accepting.

■

▸ **Lemma 35.** Given a rewrite theory $\mathcal{R}$ and $t, t' \in T_{\Sigma/E}$ such that $t \to_{\mathcal{R}}^1 t'$, there is a strategy α of the form `matchrew` P `by` x `using` $rl[Subs]\{\bar{\beta}\}$ `; match` t' such that $[\![\alpha]\!](\theta, t) = \{t'\}$.

Observe that the simpler strategy expression `all ; match` t almost satisfies the conditions required for α, but it does not guarantee termination in the presence of rules with rewriting conditions. In other words, its denotation could be $\{t', \bot\}$ in addition to $\{t'\}$. Moreover, the final test is required because there may be two rules with the same label and left-hand side but different right-hand side.

If $t \to^1_{\mathcal{R}} t'$, there must exist a (perhaps conditional) rule $rl : l \to r$ if C, a substitution σ, and a position p in t such that $t|_p = \sigma(l)$, $t' = t[p/\sigma(r)]$ and $\sigma(C)$ holds. First, we suppose C does not contain rewriting condition fragments. If $x_1, \dots, x_n$ are the variables that occur in l and C, and x is a fresh variable, the desired α is then

$$\texttt{matchrew}\ t[p/x]\ \texttt{by}\ x\ \texttt{using}\ rl[x_1 \leftarrow \sigma(x_1), \dots, x_n \leftarrow \sigma(x_n)]\ \texttt{;}\ \texttt{match}\ t'$$

This strategy forces the rl rule application to the specific position p, with the specific substitution σ, and then it is checked that the result is t'. The only possible successful execution is

$$\begin{aligned}\alpha &\to_c \text{subterm}(x : t|_p \,@\, rl[x_1 \leftarrow \sigma(x_1), \dots, x_n \leftarrow \sigma(x_n)]; t) \,@\, \texttt{match}\ t' \\ &\to_s \text{subterm}(x : \sigma(r) \,@\, \varepsilon; t) \,@\, \texttt{match}\ t' \to_c t' \,@\, \texttt{match}\ t' \to_c t' \,@\, \varepsilon\end{aligned}$$

Any other execution from α can only differ in the term after the rewrite and must fail before the test, so all possible executions are finite. Hence, $[\![\alpha]\!](\theta, t) = \{t'\}$.

If C contains rewriting conditions, strategies should be provided for these. Since the rule have been applied, for each rewriting condition rwc_l `=>` rwc_r, a sequence $t_1 t_2 \cdots t_n$ must exist with $\sigma(rwc_l) = t_1$, $\sigma(rwc_r) = t_n$ and $t_k \to^1_{\mathcal{R}} t_{k+1}$. We can apply this lemma inductively to $t_k \to^1_{\mathcal{R}} t_{k+1}$, provided that only a finite number of rewriting conditions have been computed to get $t \to^1_{\mathcal{R}} t'$. Joining all t_k transition with the concatenation operator of strategies, we get a strategy for one of the β of the `matchrew`. Doing this with all rewriting fragments, the lemma conditions hold.

■

▸ **Proposition 36.** If $S \subseteq T_\Sigma(X)$ finite and $L \subseteq S^\omega$ is a regular language and $w_i \to^1_{\mathcal{R}} w_{i+1}$ for all $w \in L$, there is a strategy α and a set $I \subseteq T_\Sigma(X)$ such that $E(\alpha, I) = L$ and the reachable states from $t \,@\, \alpha$ with optimization are finitely many for any $t \in T_\Sigma(X)$.

First, $I = \{w_0 : w \in L\}$ and there is an automaton $\mathcal{A} = (Q, S, \delta, Q_0, F)$ for the ω-regular language L. The symbols of the alphabet are states, but our strategy language is based on rules. Hence, we have to translate them.

For each pair of terms $t, t' \in S$, there is a strategy $\alpha_{t,t'}$ in the terms of Lemma 35. The transition automaton is defined $\mathcal{T} = (S \times Q, RA, \Delta, SQ_0, S \times F)$ where $RA = \{\alpha_{t,t'} \mid t, t' \in S\}$ and

$$\Delta((t,q),\alpha) = \{\ (t',q') \mid t \,@\, \alpha \to^*_{s,c} t' \,@\, \varepsilon,\ q' \in \delta(q,t')\ \} \qquad SQ_0 = \{\ (t,q) \mid t \in I,\ q \in \delta(q_0,t),\ q_0 \in Q_0\ \}$$

Note than only $\alpha_{t,t'}$ satisfies the condition on the definition of Δ.

First, we want to prove that $L = \{v \in T^\omega_{\Sigma/E} \mid v_k \,@\, w_k \to^*_{s,c} v_{k+1} \,@\, \varepsilon \text{ for all } k \in \mathbb{N}, w \in L(\mathcal{T})\}$.

- Take $w \in L(\mathcal{T})$, there is a run $\pi \in (S \times Q)^\omega$ for it in $\mathcal{T}$. Call $v \in S^\omega$ the first projection of the run $v_k = (\pi_k)_1$. By definition of Δ, $v_k \,@\, w_k \to^*_{s,c} v_{k+1} \,@\, \varepsilon$. The second projection ρ of π is an accepting run in $\mathcal{A}$ for v, because, again by definition of Δ, $\rho_{k+1} \in \delta(\rho_k, w_{k+1})$ and $\inf(\rho) \cap F$ has the same size as $\inf(\pi) \cap (S \times F)$. Hence, $v \in L$.
- Take $v \in L$. Since $\mathcal{A}$ accepts v, there is a run $\rho \in Q^\omega$ with $\inf(\rho) \cap F \neq \emptyset$. Then $\pi \in (S \times Q)^\omega$ defined by $\pi'_k = (v_k, \pi_k)$ is a valid run in $\mathcal{T}$ for the word $w \in RA^\omega$ with $w_k = \alpha_{v_k, v_{k+1}}$. In fact, $(v_{k+1}, \pi_{k+1}) \in \Delta((v_k, \pi_k), \alpha_{v_k, v_{k+1}})$ since $\pi_{k+1} \in \delta(\pi_k, v_{k+1})$ and $v_k \,@\, \alpha_{v_k, v_{k+1}} \to^*_{s,c} v_{k+1} \,@\, \varepsilon$. Then v is in the right-hand set.

Second, we will construct the strategy expression α that generates L. Since $L(\mathcal{T})$ is an ω-regular language, it can be expressed as ω-regular expression [9], which always have the form $r_1 s_1^\omega + \dots + r_n s_n^\omega$ for r_i, s_i regular expressions and $\varepsilon \notin L(s_i)$. The conversion from regular expressions to strategy expressions is almost an identity. $\emptyset$ is translated as `fail`, ε as `idle`, alternation, concatenation and iterations are the same in both languages. For each s_i, to represent s_i^ω, we define a named strategy with label f_i without arguments and defined as $T(s_i)$; f_i if T is the translation function.

Inductively, we will prove that any successful execution $t \,@\, T(r)$ for any regular or ω-regular expression r sequentially executes every strategy in a word $w \in L(r)$, and that all words in $L(r)$ can be successfully executed for some initial term. This implies, from what we have proved above, that the traces for $T(r)$ are exactly L as we want to prove. If the basic strategies $\alpha \in RA$ were more complicated, the previous would not make sense, because, how can we split an arbitrary execution to tell what sequence of strategies have been run? However, elements of RA are *atoms*, a single rule application enclosed by the `matchrew` opening and closing transitions. Only control transitions could appear between these atoms.

- If $r = \alpha_{t,t'}$ is a basic symbol, $L(r) = \{\alpha_{t,t'}\}$ and its translation is $\alpha_{t,t'}$. The only successful execution is $t \mathbin{@} \alpha_{t,t'} \to^*_{s,c} t \mathbin{@} \varepsilon$, and only if the initial term is t.
- If $r = \emptyset$, $L(r) = \emptyset$ and its translation is `fail`. No successful execution can follow from $t \mathbin{@} \texttt{fail}$.
- If $r = \varepsilon$, $L(r) = \{\varepsilon\}$ and its translation is `idle`. The only possible derivation is $t \mathbin{@} \texttt{idle} \to_c t \mathbin{@} \varepsilon$ for any $t \in T_{\Sigma/E}$, and no atom in RA is executed.
- If $e = rs$, $L(e) = L(r)L(s)$ and $T(e) = T(r)\,;T(s)$. According to Lemma 21, successful executions from $t \mathbin{@} T(r)\,;T(s)$ are concatenations of successful execution for $t \mathbin{@} T(r) \to^*_{s,c} t' \mathbin{@} \varepsilon$ and $t' \mathbin{@} T(s)$. By induction hypothesis, successful executions from $T(r)$ and $T(s)$ execute stepwise $L(r)$ and $L(s)$. So, $T(e)$ executes $L(r)$ and then $L(s)$, that is, $L(r)L(s)$.

  Moreover, given a word $v \in L(r)$ and $w \in L(s)$ there are successful executions following their steps. Hence, the composition of these is the execution that follows $vw \in L(e)$.
- If $e = r + s$, $L(e) = L(r) \cup L(s)$ and $T(e) = T(r) \,|\, T(s)$. Since

$$t \mathbin{@} T(e) \,|\, T(s) \xrightarrow{\text{c}} \begin{array}{l} t \mathbin{@} T(r) \\ t \mathbin{@} T(s) \end{array}$$

  Thus, the successful executions sequences from $t \mathbin{@} T(e)$ are the union of those for $t \mathbin{@} T(r)$ and $t \mathbin{@} T(s)$. By induction hypothesis, they stepwise execute $L(r)$ and $L(s)$, so the alternation executes $L(r) \cup L(s)$, as we wanted.
- If $e = r^*$, then $L(e) = L(r)^*$ and $T(e) = T(r)^*$. Successful executions from $t \mathbin{@} T(r)^*$ are finite iterations of successful executions for $T(r)$. Since these follow stepwise $L(r)$ by induction hypothesis, $t \mathbin{@} T(r)^*$ follows $L(r)^*$ as we want to prove.
- Now, we start with ω-regular expressions. If $e = r^\omega$, then $L(e) = L(r)^\omega$ and $T(e) = f_e$ with f_e defined as $T(r)\,;f_e$. By induction hypothesis, the atoms executed for $t \mathbin{@} T(r)$ and any term t are $L(r)$. Using the optimization, we obtain an execution loop

$$t \mathbin{@} f_e \to_c t \mathbin{@} T(r)\,;f_e \to_c t \mathbin{@} T(r)f_e \to^*_{s,c} t \mathbin{@} f_e$$

  that executes $T(r)$, so executions for f_e are $T(r)^\omega$.
- The alternation and concatenation cases for regular expressions are also valid for ω-languages (except in the first entry of the concatenation).

Finally, and since the strategy satisfies the conditions of the second statement of Proposition 4:2, the reachable states are finite.

■

A.4 Proofs for the rewriting semantics

In this section, we prove the equivalence between the denotational semantics in Section 3 and the rewriting semantics in Section 5. This is done in much the same way as the equivalence proof between the former and the operational semantics in Section A.3.

▸ **Definition.** The semantic denotation $\mathrm{dsem} : T_{\mathcal{S}(M,SM)} \to \mathcal{P}_\perp(T_\Sigma)$ of an execution state of the rewriting semantics is defined as follows:

1. $\mathrm{dsem}(\texttt{nil}) = \emptyset$.
2. $\mathrm{dsem}(\texttt{sol}(t)) = \{t\}$.
3. $\mathrm{dsem}(\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}) = [\![\alpha]\!](\mathrm{id}, t)$
4. $\mathrm{dsem}(\texttt{<}\ Q\ \texttt{;}\ \texttt{seq(}\beta\texttt{)}\ \texttt{>}) = \text{let } t \leftarrow \mathrm{dsem}(Q) : [\![\beta]\!](\mathrm{id}, t)$
5. $\mathrm{dsem}(\texttt{<}\ Q\ \texttt{;}\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>}) = \begin{cases} \text{let } t' \leftarrow \mathrm{dsem}(Q) : [\![\beta]\!](\mathrm{id}, t') & \mathrm{dsem}(Q) \neq \emptyset \\ [\![\gamma]\!](\mathrm{id}, t) & \mathrm{dsem}(Q) = \emptyset \end{cases}$

6. $\mathrm{dsem}(\texttt{< } Q \texttt{ ; chkrw(} l \texttt{ := } r \texttt{ /\ } C\texttt{, } \alpha\vec{\alpha}\texttt{, } r\texttt{, } c\texttt{) >}) =$
$\text{let } t \leftarrow \mathrm{dsem}(Q) : \bigcup_{\sigma\in\mathsf{match}(t,r)} \text{let } \sigma' \leftarrow \mathsf{check}(C,\sigma,\vec{\alpha},\mathrm{id}) : \{c(\sigma'(r))\}$
7. $\mathrm{dsem}(\texttt{< } Q \texttt{ ; mrew(} P\texttt{, } \sigma\texttt{, } c\texttt{, } x\texttt{, } x_1 \texttt{ using } \alpha_1, \dots, x_n \texttt{ using } \alpha_n\texttt{) >}) =$
$\text{let } x \leftarrow \mathrm{dsem}(Q), x_1 \leftarrow [\![\alpha_1]\!](\mathrm{id}, \sigma(x_1)), \dots, x_n \leftarrow [\![\alpha_n]\!](\mathrm{id}, \sigma(x_n)) : \{c(\sigma(P[x/t, x_1/t_1, \dots, x_n/t_n]))\}$
8. $\mathrm{dsem}(Q\,Q') = \mathrm{dsem}(Q) \cup \mathrm{dsem}(Q')$

We will make extensive use of the following statements to compose executions without explicitly mentioning them, since otherwise the text will be less readable.

▸ **Lemma 37.** For every configurations Q, Q_1, Q_2 of the operational semantics,

1. $\{Q' : Q_1 Q_2 \to^* Q'\} = \{Q_1' Q_2' : Q_1 \to^* Q_1', Q_2 \to^* Q_2'\}$.
2. The executions from the soup $Q\,S$ are exactly $(Q_{k_j}\, S_{l_j})_{j=0}^{m_1+m_2}$ for an execution $(Q_k)_{k=0}^{m_1}$ with $Q_0 = Q$ and $m_1 \in \mathbb{N}\cup\{\infty\}$, an execution $(S_k)_{k=0}^{m_2}$ with $S_0 = S$ and $m_2 \in \mathbb{N}\cup\{\infty\}$, and two monotone sequences $(k_j)_{j=0}^{m_1+m_2}$ and $(l_j)_{j=0}^{m_1+m_2}$ with $k_j + l_j = j$ for all j.
3. The sequence $(\texttt{< } Q_k \texttt{ ; } c \texttt{ >})_{k=0}^{m}$ is a valid execution iff $(Q_k)_{k=0}^{m}$ is a valid execution. Moreover, solutions may be removed from the soup Q_k and continued outside.

The rules of the rewriting semantics ensure that tasks at the same level evolve independently of each other, and their independent executions can be combined side by side in the soup according to the axioms of rewriting. Since rules can be applied inside subterms, the last statement is also valid.

■

Given an execution Π starting from $Q\,Q'$, let Π_Q be the independent execution that follows from Q as indicated above. If Π is a continuation task, Π_c will denote the execution inside the configuration and Π_{-c} the execution outside the configuration and after it is dissolved. In a strict sense, Π_c is not an execution since `sol(`t`)` states may disappear from it, but this will be immaterial.

▸ **Definition.** The call depth $\mathrm{cdepth} : T^*_{\mathcal{S}(M,SM)} \to \mathbb{N}$ of an execution in the semantics is defined as follows:

1. $\mathrm{cdepth}(\varepsilon) = 0$.
2. $\mathrm{cdepth}(\texttt{nil}) = 0$
3. $\mathrm{cdepth}(Q\,Q'\,\Pi) = \max\{\mathrm{cdepth}((Q\,\Pi)_Q), \mathrm{cdepth}((Q'\,\Pi)_{Q'})\}$.
4. $\mathrm{cdepth}(\texttt{< } \alpha \texttt{ @ } t \texttt{ >}\Pi) = \mathrm{cdepth}(\Pi)$ if α is not a strategy call or $\Pi = \varepsilon$.
5. $\mathrm{cdepth}(\texttt{< } sl(t_1, \dots, t_n) \texttt{ @ } t \texttt{ >}\Pi) = 1 + \mathrm{cdepth}(\Pi)$ if $\Pi \neq \varepsilon$.
6. $\mathrm{cdepth}(\texttt{< } Q \texttt{ ; seq(} \beta\texttt{) >}\Pi) = \max\{\mathrm{cdepth}(Q\,\Pi_{\texttt{seq}}), \mathrm{cdepth}(\Pi_{-\texttt{seq}})\}$.
7. $\mathrm{cdepth}(\texttt{< } Q \texttt{ ; ifc(} \beta\texttt{, } \gamma\texttt{, } t\texttt{) >}\Pi) = \max\{\mathrm{cdepth}(Q\,\Pi_{\texttt{ifc}}), \mathrm{cdepth}(\Pi_{-\texttt{ifc}})\}$ (if the `ifc` continuation is transformed to a `seq` continuation as indicated by the rules, $\Pi_{\texttt{ifc}}$ comprises both fragments).
8. $\mathrm{cdepth}(\texttt{< } Q \texttt{ ; chkrw(} l \texttt{ := } r \texttt{ /\ } C\texttt{, } \vec{\alpha}\texttt{, } r\texttt{, } c\texttt{) >}\Pi) = \max\{\mathrm{cdepth}(Q\,\Pi_{\texttt{chkrw}}), \mathrm{cdepth}(\Pi_{-\texttt{chkrw}})\}$.
9. $\mathrm{cdepth}(\texttt{< } Q \texttt{ ; mrew(} P\texttt{, } \sigma\texttt{, } c\texttt{, } x\texttt{, } x_1 \texttt{ using } \alpha_1, \dots, x_n \texttt{ using } \alpha_n\texttt{) >}\Pi) =$
$\max\{\mathrm{cdepth}(Q\,\Pi_{\texttt{mrew}}), \mathrm{cdepth}(\Pi_{-\texttt{mrew}})\}$.

▸ **Definition.** The relation $(\prec) \in T^2_{\mathcal{S}(M,SM)}$ is defined as the transitive closure of:

1. $\texttt{nil} \prec Q$ and $\texttt{sol(}t\texttt{)} \prec Q$ for any other Q.
2. If α is within Q and $\texttt{< } \alpha \texttt{ @ } t \texttt{ >} \neq Q$ for all $t \in T_\Sigma$, $\texttt{< } \alpha \texttt{ @ } t \texttt{ >} \prec Q$ for any $t \in T_\Sigma$.
3. $Q_1 \prec Q_1\,Q_2$ and $Q_2 \prec Q_1\,Q_2$.
4. $Q \prec \texttt{< } Q \texttt{ ; } c \texttt{ >}$ for any continuation c.

5. $\texttt{<}\ \texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}\ ;\ \texttt{seq(}\beta\texttt{)}\ \texttt{>} \prec \texttt{<}\ \alpha;\beta\ \texttt{@}\ t\ \texttt{>}$.
6. $\texttt{<}\ Q\ ;\ \texttt{seq(}\beta\texttt{)}\ \texttt{>} \prec \texttt{<}\ Q\ ;\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>}$.
7. $\texttt{<}\ \texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}\ ;\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>} \prec \texttt{<}\ \alpha\,?\,\beta : \gamma\ \texttt{@}\ t\ \texttt{>}$.
8. $\texttt{<}\ \texttt{<}\ \alpha\ \texttt{@}\ t'\ \texttt{>}\ ;\ \texttt{chkrw(}C\texttt{,}\ \vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>} \prec \texttt{<}\ rl[\alpha\vec{\alpha}]\ \texttt{@}\ t\ \texttt{>}$.
9. $\texttt{<}\ \texttt{<}\ \alpha\ \texttt{@}\ t'\ \texttt{>}\ ;\ \texttt{chkrw(}C'\texttt{,}\ \vec{\alpha}\texttt{,}\ r'\texttt{,}\ c'\texttt{)}\ \texttt{>} \prec \texttt{<}\ Q\ ;\ \texttt{chkrw(}C\texttt{,}\ \alpha\vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>}$.
10. The counterparts of (8) and (9) for the `matchrew` strategy.

▸ **Lemma 38.** The relation $(\prec) \in T^2_{\mathcal{S}(M,SM)}$ is a well-founded strict order.

We claim that all descending chains from a state Q are finite, and proceed by induction on the number of symbols in Q excluding the terms of the underlying system. Since the relation $\prec$ is the transitive closure of the axioms (1-11) in its definition, we can assume that all steps in the chain are the direct application of one of these axioms. Hence, any state of the semantics that does not match any axiom trivially satisfies the desired property, like `nil` and $\texttt{sol(}t\texttt{)}$. For any state Q, the application of the first axiom leads to these states, so all chains using it are finite. Any chain starting with a step caused by (2-4) is finite by induction hypothesis, because the left-hand side of the relation has strictly fewer symbols. In (5), the number of symbols does not decrease, but the only predecessors of the left-hand side are `nil` and $\texttt{sol(}t\texttt{)}$ by 1, $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$ by 4, and $\texttt{<}\ \gamma\ \texttt{@}\ t\ \texttt{>}$ for any substrategy γ of α or β by 2. All of them have a lower number of symbols, so all chains from the concatenation are finite. The same argument is valid for the remaining axioms, taking into account that the number of condition fragments and substrategies decreases in 9 and 11 (the implicit counterpart of 9 for the `matchrew`).

■

▸ **Lemma 39.** For any state term Q of the operational semantics, any term t, and any natural number n,

1. $t \in \text{dsem}^{(n)}(Q)$ implies $Q \to^* \texttt{sol(}t\texttt{)}\ Q'$ with a call depth of at most n.
2. $Q \to^* \texttt{sol(}t\texttt{)}\ Q'$ with a call depth of at most n implies $t \in \text{dsem}^{(n)}(Q)$.
3. $\bot \in \text{dsem}^{(n)}(Q)$ implies there is a Q' such that $Q \to^* Q'$ with a call depth of at least $n+1$ or there is an execution with infinitely many consecutive iterations on the same strategy.
4. $\bot \notin \text{dsem}^{(n)}(Q)$ implies all executions from Q and their call depths are at most n.

The proof is carried out by induction on n, which will only be explicitly visible when dealing with strategy calls, and then by induction on the terms Q via the order $\prec$. For this second induction, the simplest base cases are the states `nil` and $\texttt{sol(}t\texttt{)}$ with denotations $\emptyset$ and $\{t\}$ and no successors, which trivially satisfy all properties.

We now consider basic tasks of the form $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$, whose denotations are always $[\![\alpha]\!]^{(n)}(\text{id}, t)$. As we will later see for each case, these tasks always have a single successor $Q_1 \cdots Q_n$ with a variable number of tasks, and the denotation of $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$ is the union of the denotations of these tasks.Under these conditions, if moreover the transition is not a strategy call and successor tasks satisfy the properties, the original task satisfies them too. In effect, (1) holds because t belongs to the denotation of at least one of the successors, for which there is an execution that yields $\texttt{sol(}t\texttt{)}$, which can be extended by leaving the rest of the tasks (if any) untouched along the execution and prepending the initial state. (2) holds because tasks at the same level are completely independent, and so the $\texttt{sol(}t\texttt{)}$ term must have been reached by one of them, t belong to its denotation, and then to the denotation of the initial term. Similar arguments prove (3) since $\bot$ should be in the denotation of at least one of the tasks, and the execution with $n+1$ of call depth or infinite iterations can be extended as above; and (4), because $\bot$ cannot be contained in the denotations of any of the successor tasks, their executions are all finite, and so are their combinations extended with the initial term. Since the call depth is independent for each subtask and the first transition is not a call, the call depth of the whole path does not increase. This said, we will only focus on proving that the denotation is preserved in the first step and that the properties are satisfied by the subtasks, which usually follows from the induction hypothesis.

- For `idle`, the denotation of the task $\text{dsem}^{(n)}(\texttt{< idle @ } t \texttt{ >}) = [\![\texttt{idle}]\!]^{(n)}(\text{id}, t)$ is $\{t\}$ and its successor is `sol(`t`)`, whose denotation is also $\{t\}$ by definition. Induction hypothesis can be applied to `sol(`t`)`.

- For `fail`, the successor is `nil` and both have $\emptyset$ as denotation. Induction hypothesis can be applied to `nil`.

- For the tests, their successors are `sol(`t`)` or `nil` depending on whether the matching and condition are satisfied. In either case, the denotation is preserved, and induction hypothesis can be applied to both.

- For $\alpha \,|\, \beta$, the successor is `<` α `@` t `>` `<` β `@` t `>` and the denotations of both sides are $[\![\alpha]\!]^{(n)}(\text{id}, t) \cup [\![\beta]\!]^{(n)}(\text{id}, t)$ by definition. Induction hypothesis can be applied to both subtasks.

- For $\alpha;\beta$, the only successor of the state is `<` `<` α `@` t `>` `;` `seq(`β`)` `>` , whose denotation let $t' \leftarrow [\![\alpha]\!]^{(n)}(\text{id}, t) : [\![\beta]\!]^{(n)}(\text{id}, t')$ coincides with that of the initial state by definition. Induction hypothesis can be applied, by the fifth item in the definition of the order.

- For α`*`, the successor is `sol(`t`)` `<` α`;`α`*` `@` t `>`. The denotation of the left-hand side is $\bigcup_{k \geq 0} ([\![\alpha]\!]^{(n)})^k(\text{id}, t)$, which can be recursively expressed as $\{t\} \cup [\![\alpha;\alpha*]\!]^{(n)}(\text{id}, t)$ to match the denotation of the right-hand side. Induction hypothesis cannot be directly applied, so let consider each property one by one.

  1. Suppose $t' \in [\![\alpha*]\!]^{(n)}$. Since this is a union, there exists an $m \in \mathbb{N}$ such that $t \in ([\![\alpha*]\!]^{(n)})^m$. If $m = 0$, then $t' = t$ and the property is satisfied because of the `sol(`t`)` state. Otherwise, reasoning inductively on m, there must be a $t_m \in [\![\alpha]\!]^{(n)}$ such that $t' \in ([\![\alpha*]\!]^{(n)})^{m-1}(\text{id}, t_m)$, and then

     $$\begin{aligned} \texttt{<}\ \alpha\texttt{;}\alpha\texttt{*}\ \texttt{@}\ t\ \texttt{>} &\to \texttt{<}\ \texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}\ \texttt{;}\ \texttt{seq(}\alpha\texttt{*)}\ \texttt{>} \to^* \texttt{<}\ \texttt{sol(}t_m\texttt{)}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{*)}\ \texttt{>} \\ &\to \texttt{<}\ \alpha\texttt{*}\ \texttt{@}\ t_m\ \texttt{><}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{*)}\ \texttt{>} \to^* \texttt{sol(}t'\texttt{)}\ Q''\texttt{<}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{*)}\ \texttt{>} \end{aligned}$$

     where we have used the first property of the induction hypothesis on α to conclude `<` α `@` t `>` $\to^*$ `sol(`t_m`)` Q', and the induction hypothesis on m to obtain `<` α`*` `@` t_m `>` $\to^*$ `sol(`t'`)` Q''.
  2. Suppose `<` α`*` `@` t `>` $\to^*$ `sol(`t'`)` Q, a finite number of `seq` continuations for α`*` must have been opened and closed in the sequence of tasks leading to `sol(`t'`)`. If m is that number, we can inductively prove that $t' \in ([\![\alpha]\!]^{(n)})^m(\text{id}, t) \subseteq [\![\alpha*]\!]^{(n)}(\text{id}, t)$. If $m = 0$, $t' = t$ and we are done. Otherwise, the execution may look like in the first case, so we can conclude from the second property of the induction hypothesis on α that $t_m \in [\![\alpha]\!]^{(n)}(\text{id}, t)$ because `<` α `@` t `>` $\to^*$ `sol(`t_m`)` Q, and by the induction hypothesis on m that $t' \in ([\![\alpha]\!]^{(n)})^{m-1}(\text{id}, t_m)$. The definition of composition does the rest.
  3. If $\bot$ is contained in the denotation, two cases are possible. If $([\![\alpha]\!]^{(n)})^m(\text{id}, t) \neq \emptyset$ for all $m \in \mathbb{N}$, we can build iterations or arbitrary length with the procedure above, and there must be an infinite iteration by the Kőnig's lemma. Otherwise, $\bot \in ([\![\alpha]\!]^{(n)})^m(\text{id}, t)$ for some $m \in \mathbb{N}$ such that $\bot \in [\![\alpha]\!]^{(n)}(\text{id}, t_m)$ and $t_m \in ([\![\alpha]\!]^{(n)})^{m-1}(\text{id}, t)$. In this case, there is an execution with call depth greater than $n+1$ or with an infinite iteration from `<` α `@` t_m `>` that is reachable from `<` α`*` `@` t `>` with the construction already shown.
  4. If $\bot$ is not contained in the denotation, $\bot$ is also absent in $[\![\alpha]\!]^{(n)}(\text{id}, t_m)$ for every reachable t_m. Hence, the executions from the initial task are necessarily finite, and the call depths of the complete executions are not greater than those of the iteration, because nested calls between iterations are clearly not possible.

- For α`?`β`:`γ, the successor is `<` `<` α `@` t `>` `;` `ifc(`β`,` γ`,` t`)>` and the denotation is preserved, let $t' \leftarrow [\![\alpha]\!]^{(n)}(\text{id}, t) : [\![\beta]\!]^{(n)}(\text{id}, t')$ if $[\![\alpha]\!]^{(n)}(\text{id}, t) = \emptyset$, and $[\![\gamma]\!]^{(n)}(\text{id}, t)$ otherwise. Induction hypothesis can be applied, by the seventh point of the order definition.

- For a `matchrew`, the successor is a configuration with a `mrew`-continuation tasks for each possible match of the pattern. Checking that the definition of $\text{dsem}^{(n)}$ for this task coincides with $[\![\texttt{matchrew ...}]\!]^{(n)}$ is simply looking at them. Induction hypothesis can be applied due to the ninth item of the order definition.

- For a plain rule application $rl[\rho]$ with optional substitution ρ, the successor is a soup with `sol(`t_k`)` states, one for each possible rewrite under these conditions. The union of these terms is the denotation of the right-hand side, and this is exactly the meaning of $rl[\rho]$. Induction hypothesis can be applied too.

- For a rule application $rl[\rho]\{\vec{\alpha}\}$ with rewriting conditions, the successor is a soup of tasks with `chkrw` continuation, whose $\text{dsem}^{(n)}$ value has been defined to coincide with the denotation of the application. Induction hypothesis can be applied due to the eight point of the order definition.

- For a strategy call $sl(t_1, \dots, t_n)$, the successor is a soup with a configuration $\texttt{<}\ \sigma(\delta)\ \texttt{@}\ t\ \texttt{>}$ for every definition δ of sl and every matching substitution σ for the call term. If $n = 0$, (1) and (4) hold vacuously since the denotation can only be $\{\bot\}$, and (2) is trivially satisfied since the only possible execution with a null call depth from this term is the empty one. The third property holds, since the first step from the call term is actually an execution of call depth $0 + 1 = 1$ (even if the successor is `nil`).

  Otherwise, if $n > 0$, $[\![sl(t_1, \dots, t_n)]\!]^{(n)}(\text{id}, t) = \bigcup_{(\delta,\sigma)} [\![\delta]\!]^{(n-1)}(\sigma, t) = \bigcup_{(\delta,\sigma)} [\![\sigma(\delta)]\!]^{(n-1)}(\text{id}, t)$ by definition of the semantics and by Proposition 3:3. The denotations coincide with an application less in the second term. Hence, we can apply induction hypothesis on the call definitions, and solve (1) and (2) as usual. If $\bot$ is in the denotation of the call, it must be in the denotation of one of its executions, from which a call leaves with at least call depth n, yielding an execution with call depth $n + 1$ from the initial task. Conversely, if $\bot$ is not in any denotation, all executions from the definitions are finite with call depth bounded by $n - 1$, so their combinations are also finite and have call depth at most n once the initial step is prepended.

The case of non-atomic tasks remains. We do not explicitly argue about the existence of an infinite iteration when proving the third property, because that iteration exists in the whole execution if and only if it exists in one of the executions that are combined.

- For the union of tasks $Q_1\, Q_2$, all the four properties hold as a consequence of Lemma 37 and the induction hypothesis. The call depth of a union is the maximum of the call depths, so this does not pose any problem.

- For a state $\texttt{<}\ Q\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>}$, its denotation is let $t' \leftarrow \text{dsem}^{(n)}(Q) : [\![\alpha]\!]^{(n)}(\text{id}, t')$. If t belongs to this set, there must be some t_m such that $t_m \in \text{dsem}^{(n)}(Q)$ and $t \in [\![\alpha]\!]^{(n)}(\text{id}, t_m)$. By induction hypothesis, $Q \to^* \texttt{sol(}t_m\texttt{)}\ Q'$ with call depth at most n, and $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>} \to^* \texttt{sol(}t\texttt{)}\ Q''$ with the same call depth bound. Then,

$$\begin{aligned}\texttt{<}\ Q\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>} &\to^* \texttt{<}\ \texttt{sol(}t_m\texttt{)}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>} \to \texttt{<}\ \alpha\ \texttt{@}\ t_m\ \texttt{><}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>} \\ &\to^* \texttt{sol(}t\texttt{)}\ Q'' \texttt{<}\ Q'\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>}\end{aligned}$$

  The call depth is the maximum of the two starred transitions, and so it is bounded by n, proving the first property. The second property assumes $\texttt{<}\ Q\ \texttt{;}\ \texttt{seq(}\alpha\texttt{)}\ \texttt{>} \to^* \texttt{sol(}t\texttt{)}\ Q'''$ and due to the determinism of the semantics, this implies $Q \to^* \texttt{sol(}t_m\texttt{)} Q'$ for some t_m such that $\texttt{<}\ \alpha\ \texttt{@}\ t_m\ \texttt{>} \to^* \texttt{sol(}t\texttt{)}\ Q''$ like in the previous sequence. The call depths of these executions are necessarily below n as they are part of the complete execution, so induction hypothesis can be applied to conclude that $t_m \in \text{dsem}^{(n)}(Q)$ and $t \in [\![\alpha]\!]^{(n)}(\text{id}, t_m)$ as we wanted.

  Regarding $\bot$, the global denotation contains it iff $\bot \in \text{dsem}^{(n)}(Q)$ or $\bot \in [\![\alpha]\!]^{(n)}(\text{id}, t_m)$ for some $t_m \in \text{dsem}^{(n)}(Q)$ by definition of let. In the first case, this yields a global execution that does not pop the continuation with call depth $n + 1$. In the second case, the execution formed as above has call depth $n + 1$ because of the second starred transition. If $\bot$ is not any of those sets, the executions are concatenations of finite executions, so they are finite, and their call depths are the maximum of call depths bounded by n, so they are below n.

- For a state $\texttt{<}\ Q\ \texttt{;}\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>}$, its denotation is let $t' \leftarrow \text{dsem}^{(n)}Q : [\![\beta]\!]^{(n)}(\text{id}, t')$ if $\text{dsem}^{(n)}(Q) \neq \emptyset$, and $[\![\gamma]\!]^{(n)}(\text{id}, t)$ otherwise. In the first case, we can reason exactly as in the previous case, taking into account that the `ifc` transition is translated to a `seq` one on the first solution. For the second case, let us see that $\text{dsem}^{(n)}(Q) = \emptyset$ iff $Q \to^* \texttt{nil}$. Clearly, by induction hypothesis, $t_m \in \text{dsem}^{(n)}(Q)$ implies $Q \to^* \texttt{sol(}t_m\texttt{)} Q'$, and since `nil` cannot be rewritten to $\texttt{sol(}t_m\texttt{)}$ nor vice versa, $Q \to^* \texttt{nil}$ is impossible. Conversely, if $\text{dsem}^{(n)}(Q) = \emptyset$, then all executions from Q are finite and with call depth bounded by n. The second property of the induction hypothesis is then applicable for any execution $Q \to \texttt{sol(}t\texttt{)} Q'$, but these are impossible because $t \notin \emptyset$. Observing the rules of the operational semantics, we can see that every state has a successor except `nil` and $\texttt{sol(}t\texttt{)}$, so all executions must end in `nil`.

Using what we have just proved, if we assume t' is in the denotation and $\mathrm{dsem}^{(n)}(Q) = \emptyset$, necessarily $t' \in [\![\gamma]\!]^{(n)}(\mathrm{id}, t)$ and

$$\texttt{<}\ Q\ \texttt{;}\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>} \to^* \texttt{<}\ \texttt{nil}\ \texttt{;}\ \texttt{ifc(}\beta\texttt{,}\ \gamma\texttt{,}\ t\texttt{)}\ \texttt{>} \to \texttt{<}\ \gamma\ \texttt{@}\ t\ \texttt{>} \to^* \texttt{sol(}t'\texttt{)}\ Q'$$

Hence, the first property is satisfied. If we assume that a solution has been reached with an execution like above, this means that $\mathrm{dsem}^{(n)}(Q) = \emptyset$ and t' is in the denotation of γ, so it is in the denotation of the initial state. Similar reasons can be provided for the third and fourth properties.

- For a state $\texttt{<}\ Q\ \texttt{;}\ \texttt{chkrw(}C\texttt{,}\ \vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>}$, its denotation can be directly read from the definition. If $\vec{\alpha} = \varepsilon$, successful executions have the form

$$\begin{aligned}\texttt{<}\ Q\ \texttt{;}\ \texttt{chkrw(}C\texttt{,}\ \vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>} \to^* \texttt{<}\ \texttt{sol(}t_m\texttt{)}\ Q'\ \texttt{;}\ \texttt{chkrw(}C\texttt{,}\ \vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>}\\ \to \texttt{sol(}c(\sigma(r))\texttt{)} \texttt{<}\ Q'\ \texttt{;}\ \texttt{chkrw(}C\texttt{,}\ \vec{\alpha}\texttt{,}\ r\texttt{,}\ c\texttt{)}\ \texttt{>}\end{aligned}$$

  where σ is a matching substitution of t_m into the rewriting condition pattern. Otherwise, let $\vec{\alpha}$ be $\beta\vec{\beta}$, the $\texttt{sol(}t_m\texttt{)}$ term is replaced by another `chkrw` continuation with $\texttt{<}\ \beta\ \texttt{@}\ t\ \texttt{>}$ in the subsearch and $\vec{\beta}$ in the list of pending strategies. In both cases, induction hypothesis can be applied to prove properties (1-4). Notice that according to its definition, the call depth of the whole execution is the maximum of the subexecutions involved, so the bounds are preserved.

- For a state $\texttt{<}\ Q\ \texttt{;}\ \texttt{mrew(}P\texttt{,}\ \sigma\texttt{,}\ c\texttt{,}\ x\texttt{,}\ x_1\ \texttt{using}\ \alpha_1, \dots, x_n\ \texttt{using}\ \alpha_n\texttt{)}\ \texttt{>}$ the proof is very similar to the previous case.

■

▸ **Lemma 40.** For any execution state Q, $\mathrm{dsem}(Q) = \sup\{\mathrm{dsem}^{(n)}(Q) : n \in \mathbb{N}\}$. Moreover, $\bigcup_{n\in\mathbb{N}} \mathrm{dsem}^{(n)}(Q) \cap T_\Sigma(X) = \mathrm{dsem}(Q) \cap T_\Sigma(X)$.

The statement follows from the fact that $\mathrm{dsem}^{(n)}$ is the definition of dsem with $[\![\alpha]\!]$ replaced by $[\![\alpha]\!]^{(n)}$, that $\sup [\![\alpha]\!]^{(n)} = [\![\alpha]\!]$, and that those definitions are continuous. The consequence follows from the properties of the order and its supremum.

■

▸ **Proposition 41.** For any state Q of the operational semantics,

$$t \in \mathrm{dsem}(Q) \iff \exists\, Q' \quad Q \to^* \texttt{sol(}t\texttt{)}\ Q'.$$

Moreover, $\bot \in \mathrm{dsem}(Q)$ iff there is an infinite execution from Q.

Each of the implications in this statement easily follows from an item of Lemma 39 using Lemma 40 extensively.

1. If $t \in \mathrm{dsem}(Q)$, then $t \in \mathrm{dsem}^{(n)}(Q)$ for some $n \in \mathbb{N}$, and so $Q \to^* \texttt{sol(}t\texttt{)}\ Q'$ for some Q' by the first item of the lemma.
2. The call depth of a fixed path $Q \to^* \texttt{sol(}t\texttt{)}\ Q'$ is a fixed number n, and so the second item can be used to conclude $t \in \mathrm{dsem}^{(n)}(Q)$ and by containment $t \in \mathrm{dsem}(Q)$.
3. If $\bot \in \mathrm{dsem}(Q)$, $\bot \in \mathrm{dsem}^{(n)}(Q)$ for all $n \in \mathbb{N}$ by the properties of the supremum. Hence, there are paths of arbitrary call depth or a path with infinitely many iterations, which is necessarily infinite. In the first case, since $\to$ is finitary and by the Kőnig lemma, there is an infinite execution from Q.
4. If $\bot \notin \mathrm{dsem}(Q)$, there must be a number n such that $\bot \notin \mathrm{dsem}^{(n)}(Q)$. The fourth item of the lemma then asserts that all executions from Q are finite.

■

▸ **Theorem 42.** For any terms t, t' and strategy expression α,

$$t' \in [\![\alpha]\!](\mathrm{id}, t) \iff \exists\, Q' \quad \texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>} \to^* \texttt{sol(}t'\texttt{)}\ Q'.$$

Moreover, $\bot \in [\![\alpha]\!](\mathrm{id}, t)$ iff there is an infinite execution from $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$.

This is a direct corollary of Proposition 41, where Q is replaced by $\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>}$, and the set $\mathrm{dsem}(\texttt{<}\ \alpha\ \texttt{@}\ t\ \texttt{>})$ is legitimately replaced by its definition $[\![\alpha]\!](\mathrm{id}, t)$.

■

Acknowledgements

Research partially supported by the Spanish Ministry of Science and Innovation through the projects TRACES (TIN2015-67522-C3-3-R) and ProCode (PID2019-108528RB-C22), and by the Ministry of Universities via the grant FPU17/02319.

References

[1] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. *Maude Manual v3.1*. October 2020.

[2] S. Eker, N. Martí-Oliet, J. Meseguer, and A. Verdejo. Deduction, strategies, and rewriting. In M. Archer, T. B. de la Tour, and C. Muñoz, editors, *Proceedings of the 6th International Workshop on Strategies in Automated Deduction, STRATEGIES 2006, Seattle, WA, USA, August 16, 2006*, volume 174(11) of *Electronic Notes in Theoretical Computer Science*, pages 3–25. Elsevier, 2007.

[3] J. Gaoubault-Larrecq. *Non-Hausdorff Topology and Domain Theory*. New Mathematical Monographs. Cambridge University Press, 2013.

[4] C. Kirchner, F. Kirchner, and H. Kirchner. Strategic computation and deduction. In C. Benzmüller, C. E. Brown, J. Siekmann, and R. Statman, editors, *Reasoning in Simple Type Theory. Festchrift in Honour of Peter B. Andrews on His 70th Birthday*. Volume 17, Studies in Logic and the Foundations of Mathematics, pages 339–364. College Publications, 2008.

[5] H. Kirchner. Rewriting strategies and strategic rewrite programs. In N. Martí-Oliet, P. C. Ölveczky, and C. L. Talcott, editors, *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2015. ISBN: 978-3-319-23164-8.

[6] D. Kőnig. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3:121–130, 1927.

[7] N. Martí-Oliet, J. Meseguer, and A. Verdejo. A rewriting semantics for Maude strategies. In G. Roşu, editor, *Proceedings of the Seventh International Workshop on Rewriting Logic and its Applications, WRLA 2008, Budapest, Hungary, March 29-30, 2008*, volume 238(3) of *Electronic Notes in Theoretical Computer Science*, pages 227–247. Elsevier, 2009.

[8] N. Martí-Oliet, J. Meseguer, and A. Verdejo. Towards a strategy language for Maude. In N. Martí-Oliet, editor, *Proceedings of the Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27-April 4, 2004*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier, 2004.

[9] D. Parker. ω-regular languages. Technical report, Department of Computer Science. University of Oxford, 2011.

[10] R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Model checking strategy-controlled rewriting systems. In H. Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[11] R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Strategies, model checking and branching-time properties in Maude. In S. Escobar and N. Martí-Oliet, editors, *Rewriting Logic and Its Applications - 13th International Workshop, WRLA 2020, Virtual Event, October 20-22, 2020, Revised Selected Papers*, volume 12328 of *Lecture Notes in Computer Science*, pages 156–175. Springer, 2020. ISBN: 978-3-030-63595-4.

[12] R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Strategies, model checking and branching-time properties in Maude. *J. Log. Algebr. Methods Program.*, 123:1–28, 2021.

[13] G. Winskel. *The Formal Semantics of Programming Languages. An Introduction*. Foundations of Computing. The MIT Press, 1994.