Structuring the Synthesis of Heap-Manipulating Programs

NADIA POLIKARPOVA, UCSD, USA
ILYA SERGEY, University College London, UK

Introduction

$\{x \mapsto a * y \mapsto b\}$ **void** swap(**loc** x, **loc** y) $\{x \mapsto b * y \mapsto a\}$

Intérêt : Faire avancer l'état de l'art en matière de synthèse de programmes qui manipulent des pointeurs à partir de spécifications fonctionelles formelles.

Idée Clé : Utiliser la logique de séparation.

Contributions : Synthetic Separation Logic un systeme de preuve. Et SuSLik leur synthétiseur

Spécifications pour la Synthèse

On utilise ici des tas symboliques.

$$\Sigma; \Gamma; \{\mathcal{P}\} \rightsquigarrow \{\mathcal{Q}\} | c$$

- Γ : environnement
- Σ : contexte
- $\mathcal{P}$,ϕ,P : précondition, ses parties pure et spatiale
- $\mathcal{Q}$,ψ,Q : postcondition, ses parties pure et spatiale
- $GV(\Gamma, \mathcal{P}, \mathcal{Q}) = Vars(\mathcal{P}) \backslash \Gamma$
- $EV(\Gamma, \mathcal{P}, \mathcal{Q}) = Vars(\mathcal{Q}) \backslash (\Gamma \cup Vars(\mathcal{P})$

Règles d'Inférence Basiques

Un exemple

$$\textsc{Emp}\quad \frac{\mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) = \emptyset \qquad \phi \Rightarrow \psi}{\Gamma;\ \{\phi; \mathsf{emp}\} \leadsto \{\psi; \mathsf{emp}\} \mid \mathsf{skip}}$$

$$\textsc{Read}\quad \frac{a \in \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad y \notin \mathsf{Vars}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma \cup \{y\};\ [y/a]\{\phi; \langle x, \iota\rangle \mapsto a * P\} \leadsto [y/a]\{Q\} \mid c}{\Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto a * P\} \leadsto \{Q\} \mid \mathbf{let}\ y = *(x + \iota); c}$$

$$\textsc{Write}\quad \frac{\mathsf{Vars}(e) \subseteq \Gamma \qquad \Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto e * P\} \leadsto \{\psi; \langle x, \iota\rangle \mapsto e * Q\} \mid c}{\Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto e' * P\} \leadsto \{\psi; \langle x, \iota\rangle \mapsto e * Q\} \mid *(x + \iota) = e; c}$$

$$\textsc{Frame}\quad \frac{\mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \cap \mathsf{Vars}(R) = \emptyset \qquad \Gamma;\ \{\phi; P\} \leadsto \{\psi; Q\} \mid c}{\Gamma;\ \{\phi; P * R\} \leadsto \{\psi; Q * R\} \mid c}$$

Fig. 1. Simplified basic rules of SSL.

$$\frac{}{\{x, y, \mathsf{a2}, \mathsf{b2}\};\ \{\mathsf{emp}\} \leadsto \{\mathsf{emp}\}}\ \textsc{Emp}\ \text{with}\ c_7 = \mathsf{skip}$$

$$c_6 = c_7$$

$$\frac{\vdots}{\{x, y, \mathsf{a2}, \mathsf{b2}\};\ \{y \mapsto \mathsf{a2}\} \leadsto \{y \mapsto \mathsf{a2}\} \mid c_6}\ \textsc{Frame}$$

$$c_5 = *y = \mathsf{a2}; c_6$$

$$\frac{\vdots}{\{x, y, \mathsf{a2}, \mathsf{b2}\};\ \{y \mapsto \mathsf{b2}\} \leadsto \{y \mapsto \mathsf{a2}\} \mid c_5}\ \textsc{Write}$$

$$c_4 = c_5$$

$$\frac{\vdots}{\{x, y, \mathsf{a2}, \mathsf{b2}\};\ \{x \mapsto \mathsf{b2} * y \mapsto \mathsf{b2}\} \leadsto \{x \mapsto \mathsf{b2} * y \mapsto \mathsf{a2}\} \mid c_4}\ \textsc{Frame}$$

$$c_3 = *x = \mathsf{b2}; c_4$$

$$\frac{\vdots}{\{x, y, \mathsf{a2}, \mathsf{b2}\};\ \{x \mapsto \mathsf{a2} * y \mapsto \mathsf{b2}\} \leadsto \{x \mapsto \mathsf{b2} * y \mapsto \mathsf{a2}\} \mid c_3}\ \textsc{Write}$$

$$c_2 = \mathbf{let}\ \mathsf{b2} = *y; c_3$$

$$\frac{\vdots}{\{x, y, \mathsf{a2}\};\ \{x \mapsto \mathsf{a2} * y \mapsto b\} \leadsto \{x \mapsto b * y \mapsto \mathsf{a2}\} \mid c_2}\ \textsc{Read}$$

$$c_1 = \mathbf{let}\ \mathsf{a2} = *x; c_2$$

$$\frac{\vdots}{\{x, y\};\ \{x \mapsto a * y \mapsto b\} \leadsto \{x \mapsto b * y \mapsto a\} \mid c_1}\ \textsc{Read}$$

Fig. 2. Derivation of swap(x,y) as c_1.

Règles d'Inférence Basiques

EMP terminale, parties spatiales vide, $EV = \emptyset$, $\phi \implies \psi$
skip

READ assigne la valeur d'une GV a une nouvelle variable de programme et substitue toutes les occurences.
let b = *x

WRITE assigne l'évaluation d'une expression e à une case mémoire.
*x = b

FRAME Enlève une partie spatiale commune à ϕ et ψ, si cela ne crée pas de variable existentielle.
skip

Unification Spatiale et Backtrack

$\{x \mapsto 239 * y \mapsto 30\}$ `void pick(loc x, loc y)` $\{x \mapsto z * y \mapsto z\}$

Avec la substitution $z \mapsto 239$, on unfie z et 239.

$$\{x \mapsto 239 * y \mapsto 30\} \rightsquigarrow \{x \mapsto 239 * y \mapsto 239\}.$$

Introduit du déterminisme et peut alors nécessiter du backtracking !

$\{x \mapsto a * y \mapsto b\}$ `void notSure(loc x, loc y)` $\{x \mapsto c * c \mapsto 0\}$

Si on lit x dans une variable a_2, on a le but (impossible)

$$\{x, y, a_2\}\{y \mapsto b\} \rightsquigarrow \{a_2 \mapsto 0\}.$$

Raisonner sur les contraintes pures

Précondition

$\{a = x \wedge y = a; x \mapsto y * y \mapsto z\}$ **void** urk(**loc** x, **loc** y) $\{\text{true}; y \mapsto a * x \mapsto y\}$

Deux variables universelles égales, x et y, on substitue.

$$\{x, y\}\{y \mapsto x * x \mapsto z\} \rightsquigarrow \{x \mapsto x * x \mapsto x\}.$$

Ici, mène à quelque chose d'impossible $\implies$ règle d'inconsistence !

Raisonner sur les contraintes pures

Les règles

$$\textsc{SubstLeft}\quad \frac{\phi \Rightarrow x = y \qquad \Gamma; [y/x]\{\phi; P\} \rightsquigarrow [y/x]\{Q\} \,|\, c}{\Gamma; \{\phi; P\} \rightsquigarrow \{Q\} \,|\, c}$$

$$\textsc{StarPartial}\quad \frac{x + \iota \neq y + \iota' \notin \phi \qquad \phi' = \phi \wedge (x + \iota \neq y + \iota') \qquad \Gamma; \left\{\phi'; \langle x, \iota\rangle \mapsto e * \left\langle y, \iota'\right\rangle \mapsto e' * P\right\} \rightsquigarrow \{Q\} \Big|\, c}{\Gamma; \left\{\phi; \langle x, \iota\rangle \mapsto e * \left\langle y, \iota'\right\rangle \mapsto e' * P\right\} \rightsquigarrow \{Q\} \Big|\, c}$$

$$\textsc{Inconsistency}\quad \frac{\phi \Rightarrow \bot}{\Gamma; \{\phi; P\} \rightsquigarrow \{Q\} \,|\, \mathsf{error}}$$

$$\textsc{SubstRight}\quad \frac{x \in \mathrm{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Sigma; \Gamma; \{\mathcal{P}\} \rightsquigarrow [e/x]\{\psi, Q\} \,|\, c}{\Sigma; \Gamma; \{\mathcal{P}\} \rightsquigarrow \{\psi \wedge x = e; Q\} \,|\, c}$$

$$\textsc{Pick}\quad \frac{y \in \mathrm{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \mathrm{Vars}(e) \in \Gamma \cup \mathrm{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma; \{\phi; P\} \rightsquigarrow [e/y]\{\psi; Q\} \,|\, c}{\Gamma; \{\phi; P\} \rightsquigarrow \{\psi; Q\} \,|\, c}$$

$$\textsc{UnifyPure}\quad \frac{[\sigma]\psi' = \phi' \qquad \emptyset \neq \mathrm{dom}(\sigma) \subseteq \mathrm{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma; \{\mathcal{P}\} \rightsquigarrow [\sigma]\{\mathcal{Q}\} \,|\, c}{\Gamma; \{\phi \wedge \phi'; P\} \rightsquigarrow \{\psi \wedge \psi'; Q\} \Big|\, c}$$

Fig. 4. Selected SSL rules for reasoning with pure constraints in the synthesis goal.

Synthèse pour prédicats inductifs

Mémoire dynamique

$$\begin{aligned}\mathsf{lseg}(x, y, S) \triangleq\ & x = y \land \{S = \emptyset; \mathsf{emp}\}\\ & x \neq y \land \{S = \{v\} \cup S_1; [x, 2] * x \mapsto v * \langle x, 1\rangle \mapsto nxt * \mathsf{lseg}(nxt, y, S_1)\}\end{aligned}$$

Définition d'une liste chaînée dont le premier pointeur est x, le dernier y et les éléments sont ceux de S.

On étend notre langage avec

- Des prédicats inductifs.
- Des blocs de mémoire (et ajout de règles ALLOC et FREE).

Synthèse pour prédicats inductifs

Induction

$$\{\mathrm{lseg}(x, 0, S)\}\ \texttt{void listfree(loc x)}\ \{\mathrm{emp}\}$$

On veut synthétiser la fonction pour libérer une liste chaînée.

Une règle INDUCTION qui

- ajoute la fonction au contexte (permettre les appels récursifs) ;
- ajoute une étiquette à la fonction (utile pour la terminaison).

$$\Sigma_1 \triangleq \Sigma, \texttt{listfree}(x') : \{\mathrm{lseg}^1(x', 0, S')\}\{\mathrm{emp}\}$$

Déroulement de prédicat

Après la règle Induction, une règle Open qui *unfold* la définition du prédicat et génrère deux buts à résoudre.

$(i)\ \Sigma_1; \{x\}\,; \{x = 0 \wedge S = \emptyset; \mathsf{emp}\} \leadsto \{\mathsf{emp}\}$

$(ii)\ \Sigma_1; \{x\}\,; \left\{x \neq 0 \wedge S = \{v\} \cup S_1; [x, 2] * x \mapsto v * \langle x, 1\rangle \mapsto nxt * \mathsf{lseg}^1(nxt, y, S_1)\right\} \leadsto \{\mathsf{emp}\}$

c_1 et c_2 programmes pour (1) et (2), programme final

$$\mathrm{if}(x = 0)\{c_1\}\,\mathrm{else}\{c_2\}.$$

Synthèse pour prédicats inductifs

Retour à l'étiquette de niveau

But : éviter les dérivations infinies (ici donne programmes qui ne terminent pas en s'appellant eux-mêmes).

Idéalement : prédicats bien-fondés et applications récursives sur tas strictement plus petits.

Méthode : avec les tags, on empêche la fonction de s'appeler sur le même tas en post-condition.

Synthèse pour prédicats inductifs

Déroulement dans la postcondition

Si un prédicat inductif dans la postcondition.

Une règle CLOSE.

- Choisit non déterministiquement la partie du prédicat à satisfaire.
- Met à jour la postconditions avec la postcondition de la partie choisie.
- Incrémente l'étiquette.

Permettre l'appel de procèdure

Enlévement de l'appel

$\{r \mapsto x * \mathsf{lseg}(x, 0, S)\}$ `void listcopy(loc r)` $\{r \mapsto y * \mathsf{lseg}(x, 0, S) * \mathsf{lseg}(y, 0, S)\}$

Induction ajoute cette fonction à l'environnement.

`void listcopy(loc r')` : $\{r' \mapsto x' * \mathsf{lseg}^1(x', 0, S')\}\{r' \mapsto y' * \mathsf{lseg}^1(x', 0, S') * \mathsf{lseg}^1(y', 0, S')\}$

Open (plus d'autres règles) créent ce but.

$$\{x, r, \mathsf{x2}, \mathsf{v2}, \mathsf{nxt2}\}\,;\ \begin{array}{l}\left\{S = \{\mathsf{v2}\} \mathbin{\dot\cup} S_1 \wedge \mathsf{x2} \neq 0;\ r \mapsto \mathsf{x2} * \mathsf{lseg}^1(\mathsf{nxt2}, 0, S_1)\right\} \leadsto \\ \left\{r \mapsto y * \mathsf{lseg}^1(\mathsf{nxt2}, 0, S_2) * \mathsf{lseg}^0(y, 0, S)\right\}\end{array}$$

Peut pas utiliser CALL (il faut $r \mapsto nxt2$).

Solution : ajout d'une règle AbduceCall qui « prépare »le tas. Essaie d'unifier les préconditions de notre but et celles de la fonction qu'on veut appeler.

Synthetic Separation Logic

Variable	x, y		Alpha-numeric identifiers
Value	d		Theory-specific atoms
Offset	ι		Non-negative integers
Expression	e	$::=$	$d \mid x \mid e = e \mid e \wedge e \mid \neg e \mid \ldots$
Command	c	$::=$	$\textbf{let}\ x = *(x + \iota) \mid *(x + \iota) = e \mid$ $\texttt{skip} \mid \texttt{error} \mid \texttt{magic} \mid$ $\textbf{if}\ (e)\ \{c\}\ \textbf{else}\ \{c\} \mid f(\overline{e_i}) \mid c; c$
Type	t	$::=$	$\textbf{loc} \mid \textbf{int} \mid \textbf{bool} \mid \textbf{set}$
Fun. dict.	Δ	$::=$	$\epsilon \mid \Delta, f\ (\overline{t_i\ x_i})\ \{\ c\ \}$

Fig. 10. Programming language grammar.

Pure assertion	ϕ, ψ, ξ, χ	$::=$	e
Symbolc heap	P, Q, R	$::=$	$\mathsf{emp} \mid \langle e, \iota \rangle \mapsto e \mid$ $[x, n] \mid p(\overline{x_i}) \mid P * Q$
Assertion	$\mathcal{P}, \mathcal{Q}$	$::=$	$\{\phi, P\}$
Heap predicate	$\mathcal{D}$	$::=$	$p\ (\overline{x_i})\ \overline{\langle \xi_j, \{\chi_j, R_j\} \rangle}$
Function spec	$\mathcal{F}$	$::=$	$f\ (\overline{x_i}) : \{\mathcal{P}\}\{\mathcal{Q}\}$
Environment	Γ	$:=$	$\epsilon \mid \Gamma, x$
Context	Σ	$:=$	$\epsilon \mid \Sigma, \mathcal{D} \mid \Sigma, \mathcal{F}$

Fig. 11. SSL assertion syntax.

Synthetic Separation Logic

Induction

$$\dfrac{\begin{array}{c} f \triangleq \text{goal's name} \\ \overline{x_i} \triangleq \text{goal's formals} \\ P_f \triangleq p^1(\overline{y_i}) * \lceil P \rceil \qquad Q_f \triangleq \lceil Q \rceil \\ \mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f\}\{\psi_f; Q_f\} \\ \Sigma,\ \mathcal{F}; \Gamma;\ \{\phi; p^0(\overline{y_i}) * P\} \rightsquigarrow \{Q\} \,\middle|\, c \end{array}}{\Sigma; \Gamma;\ \{\phi; p^0(\overline{y_i}) * P\} \rightsquigarrow \{Q\} \,\middle|\, c}$$

Emp

$$\dfrac{\mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) = \emptyset \qquad \phi \Rightarrow \psi}{\Gamma;\ \{\phi; \mathsf{emp}\} \rightsquigarrow \{\psi; \mathsf{emp}\} \mid \mathsf{skip}}$$

NullNotLVal

$$\dfrac{\begin{array}{c} x \neq 0 \notin \phi \qquad \phi' \triangleq \phi \wedge x \neq 0 \\ \Sigma; \Gamma;\ \{\phi'; \langle x, \iota\rangle \mapsto e * P\} \rightsquigarrow \{Q\} \,\middle|\, c \end{array}}{\Sigma; \Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto e * P\} \rightsquigarrow \{Q\} \,\middle|\, c}$$

Inconsistency

$$\dfrac{\phi \Rightarrow \bot}{\Gamma;\ \{\phi; P\} \rightsquigarrow \{Q\} \mid \mathsf{error}}$$

SubstLeft

$$\dfrac{\begin{array}{c} \phi \Rightarrow x = y \\ \Gamma;\ [y/x]\{\phi; P\} \rightsquigarrow [y/x]\{Q\} \mid c \end{array}}{\Gamma;\ \{\phi; P\} \rightsquigarrow \{Q\} \mid c}$$

StarPartial

$$\dfrac{\begin{array}{c} x + \iota \neq y + \iota' \notin \phi \qquad \phi' \triangleq \phi \wedge (x + \iota \neq y + \iota') \\ \Sigma; \Gamma;\ \{\phi'; \langle x, \iota\rangle \mapsto e * \langle y, \iota'\rangle \mapsto e' * P\} \rightsquigarrow \{Q\} \,\middle|\, c \end{array}}{\Sigma; \Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto e * \langle y, \iota'\rangle \mapsto e' * P\} \rightsquigarrow \{Q\} \,\middle|\, c}$$

Read

$$\dfrac{\begin{array}{c} a \in \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad y \notin \mathsf{Vars}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Gamma \cup \{y\};\ [y/a]\{\phi; \langle x, \iota\rangle \mapsto a * P\} \rightsquigarrow [y/a]\{Q\} \mid c \end{array}}{\Sigma; \Gamma;\ \{\phi; \langle x, \iota\rangle \mapsto a * P\} \rightsquigarrow \{Q\} \mid \mathbf{let}\ y = *(x + \iota); c}$$

Open

$$\dfrac{\begin{array}{c} \mathcal{D} \triangleq p(\overline{x_i})\overline{\langle \xi_j, \{\chi_j, R_j\}\rangle}_{j \in 1\ldots N} \in \Sigma \\ \ell < \mathsf{MaxUnfold} \quad \sigma \triangleq [\overline{x_i \mapsto y_i}] \quad \mathsf{Vars}(\overline{y_i}) \subseteq \Gamma \\ \phi_j \triangleq \phi \wedge [\sigma]\xi_j \wedge [\sigma]\chi_j \qquad P_j \triangleq \lceil [\sigma]R_j \rceil^{\ell+1} * \lceil P \rceil \\ \forall j \in 1\ldots N, \quad \Sigma; \Gamma;\ \{\phi_j; P_j\} \rightsquigarrow \{Q\} \,\middle|\, c_j \\ c \triangleq \mathbf{if}\ ([\sigma]\xi_1)\ \{c_1\}\ \mathbf{else}\ \{\mathbf{if}\ ([\sigma]\xi_2) \ldots \mathbf{else}\ \{c_N\}\} \end{array}}{\Sigma; \Gamma;\ \left\{\phi; P * p^{\ell}(\overline{y_i})\right\} \rightsquigarrow \{Q\} \,\middle|\, c}$$

Close

$$\dfrac{\begin{array}{c} \mathcal{D} \triangleq p(\overline{x_i})\overline{\langle \xi_j, \{\chi_j, R_j\}\rangle}_{j \in 1\ldots N} \in \Sigma \\ \ell < \mathsf{MaxUnfold} \qquad \sigma \triangleq [\overline{x_i \mapsto y_i}] \\ \text{for some } k,\ 1 \leq k \leq N \qquad R' \triangleq \lceil [\sigma]R_k \rceil^{\ell+1} \\ \Sigma; \Gamma;\ \{\mathcal{P}\} \rightsquigarrow \{\psi \wedge [\sigma]\xi_k \wedge [\sigma]\chi_k; Q * R'\} \,\middle|\, c \end{array}}{\Sigma; \Gamma;\ \{\mathcal{P}\} \rightsquigarrow \left\{\psi; Q * p^{\ell}(\overline{y_i})\right\} \,\middle|\, c}$$

Synthetic Separation Logic

AbduceCall

$$\frac{\begin{array}{c}\mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f * F_f\}\{\psi_f; Q_f\} \in \Sigma \\ F_f \text{ has no predicate instances} \qquad [\sigma]P_f = P \\ F_f \neq \mathsf{emp} \qquad F' \triangleq [\sigma]F_f \qquad \Sigma; \Gamma; \{\phi; F\} \rightsquigarrow \{\phi; F'\} \mid c_1 \\ \Sigma; \Gamma; \{\phi; P * F' * R\} \rightsquigarrow \{Q\} \mid c_2\end{array}}{\Sigma; \Gamma; \{\phi; P * F * R\} \rightsquigarrow \{Q\} \mid c_1; c_2}$$

Call

$$\frac{\begin{array}{c}\mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f\}\{\psi_f; Q_f\} \in \Sigma \\ R =^{\ell} [\sigma]P_f \qquad \phi \Rightarrow [\sigma]\phi_f \\ \phi' \triangleq [\sigma]\psi_f \qquad R' \triangleq \lceil [\sigma]Q_f \rceil \qquad \overline{e_i} = [\sigma]\overline{x_i} \\ \mathsf{Vars}(\overline{e_i}) \subseteq \Gamma \qquad \Sigma; \Gamma; \{\phi \wedge \phi'; P * R'\} \rightsquigarrow \{Q\} \mid c\end{array}}{\Sigma; \Gamma; \{\phi; P * R\} \rightsquigarrow \{Q\} \mid f(\overline{e_i}); c}$$

Alloc

$$\frac{\begin{array}{c}R = [z, n] * \mathop{\ast}_{0 \le i \le n} (\langle z, i \rangle \mapsto e_i) \qquad z \in \mathsf{EV}(\Gamma, \mathcal{P}, Q) \\ (\{y\} \cup \{\overline{t_i}\}) \cap \mathsf{Vars}(\Gamma, \mathcal{P}, Q) = \emptyset \\ R' \triangleq [y, n] * \mathop{\ast}_{0 \le i \le n} (\langle y, i \rangle \mapsto t_i) \\ \Sigma; \Gamma; \{\phi; P * R'\} \rightsquigarrow \{\psi; Q * R\} \mid c\end{array}}{\Sigma; \Gamma; \{\phi; P\} \rightsquigarrow \{\psi; Q * R\} \mid \mathbf{let}\ y = \mathtt{malloc}(n); c}$$

Free

$$\frac{\begin{array}{c}R = [x, n] * \mathop{\ast}_{0 \le i \le n} (\langle x, i \rangle \mapsto e_i) \\ \mathsf{Vars}(\{x\} \cup \{\overline{e_i}\}) \subseteq \Gamma \qquad \Sigma; \Gamma; \{\phi; P\} \rightsquigarrow \{Q\} \mid c\end{array}}{\Sigma; \Gamma; \{\phi; P * R\} \rightsquigarrow \{Q\} \mid \mathsf{free}(n); c}$$

Write

$$\frac{\mathsf{Vars}(e) \subseteq \Gamma \qquad \Gamma; \{\phi; \langle x, \iota \rangle \mapsto e * P\} \rightsquigarrow \{\psi; \langle x, \iota \rangle \mapsto e * Q\} \mid c}{\Gamma; \{\phi; \langle x, \iota \rangle \mapsto e' * P\} \rightsquigarrow \{\psi; \langle x, \iota \rangle \mapsto e * Q\} \mid *(x + \iota) = e; c}$$

Synthetic Separation Logic

$$\textsc{UnifyHeaps}\quad \frac{[\sigma]R' = R \qquad \boxed{\mathsf{frameable}}\ (R') \qquad \emptyset \neq \mathrm{dom}(\sigma) \subseteq \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma;\ \{P * R\} \leadsto [\sigma]\{\psi; Q * R'\} \mid c}{\Gamma;\ \{\phi; P * R\} \leadsto \{\psi; Q * R'\} \mid c}$$

$$\textsc{Frame}\quad \frac{\mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \cap \mathsf{Vars}(R) = \emptyset \qquad \boxed{\mathsf{frameable}}\ (R') \qquad \Gamma;\ \{\phi; P\} \leadsto \{\psi; Q\} \mid c}{\Gamma;\ \{\phi; P * R\} \leadsto \{\psi; Q * R\} \mid c}$$

$$\textsc{Pick}\quad \frac{y \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \mathsf{Vars}(e) \in \Gamma \cup \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma;\ \{\phi; P\} \leadsto [e/y]\{\psi; Q\} \mid c}{\Gamma;\ \{\phi; P\} \leadsto \{\psi; Q\} \mid c}$$

$$\textsc{UnifyPure}\quad \frac{[\sigma]\psi' = \phi' \qquad \emptyset \neq \mathrm{dom}(\sigma) \subseteq \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Gamma;\ \{\mathcal{P}\} \leadsto [\sigma]\{\mathcal{Q}\} \mid c}{\Gamma;\ \{\phi \wedge \phi'; P\} \leadsto \{\psi \wedge \psi'; Q\} \mid c}$$

$$\textsc{SubstRight}\quad \frac{x \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad \Sigma; \Gamma;\ \{\mathcal{P}\} \leadsto [e/x]\{\psi, \mathcal{Q}\} \mid c}{\Sigma; \Gamma;\ \{\mathcal{P}\} \leadsto \{\psi \wedge x = e; \mathcal{Q}\} \mid c}$$

Garanties Formelles

La validité pour la partie SL est assez similaire au cas plus classique.

- $\langle h, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; \mathrm{emp}\}$ *iff* $[\![\phi]\!]_s = \text{true}$ and $\mathrm{dom}\,(h) = \emptyset$.
- $\langle h, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; [x, n]\}$ *iff* $[\![\phi]\!]_s = \text{true}$ and $\mathrm{dom}\,(h) = \emptyset$.
- $\langle h, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; \langle e_1, \iota\rangle \mapsto e_2\}$ *iff* $[\![\phi]\!]_s = \text{true}$ and $\mathrm{dom}\,(h) = [\![e_1]\!]_s + \iota$ and $h([\![e_1]\!]_s + \iota) = [\![e_2]\!]_s$.
- $\langle h, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; P_1 * P_2\}$ *iff* $\exists h_1, h_2, h = h_1 \uplus h_2$ and $\langle h_1, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; P_1\}$ and $\langle h_2, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; P_2\}$.
- $\langle h, s\rangle \vDash_{\mathcal{I}}^{\Sigma} \{\phi; p(\overline{x_i})\}$ *iff* $[\![\phi]\!]_s = \text{true}$ and $\mathcal{D} \triangleq p(\overline{x_i})\overline{\langle \xi_j, \{\chi_j, R_j\}\rangle} \in \Sigma$ and $\left\langle h, \overline{[\![x_i]\!]_s}\right\rangle \in \mathcal{I}(\mathcal{D})$.

Garanties Formelles

Definition 3.1 (Sized validity). We say a specification $\Sigma; \Gamma; \{\mathcal{P}\}\ c\ \{\mathcal{Q}\}$ is *n-valid wrt.* the function dictionary Δ whenever for any h, h', s, s' such that

- $|h| \leq n$,
- $\Delta; \langle h, (c, s) \cdot \epsilon \rangle \rightsquigarrow^* \langle h', (\texttt{skip}, s') \cdot \epsilon \rangle$, and
- $\mathrm{dom}\,(s) = \Gamma$ and $\exists \sigma_{\mathrm{gv}} = [\overline{x_i \mapsto d_i}]_{x_i \in \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q})}$ such that $\langle h, s \rangle \vDash_{\mathcal{I}}^{\Sigma} [\sigma_{\mathrm{gv}}]\mathcal{P}$,

it is the case that $\exists \sigma_{\mathrm{ev}} = [\overline{y_j \mapsto d_j}]_{y_j \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q})}$, such that $\langle h', s' \rangle \vDash_{\mathcal{I}}^{\Sigma} [\sigma_{\mathrm{ev}} \uplus \sigma_{\mathrm{gv}}]\mathcal{Q}$

On définit une correction vis à vis de la pré et post condition mais seulement pour des tas de taille n.

Garanties Formelles

Definition 3.2 (Coherence). A dictionary Δ is *n-coherent wrt.* a context Σ ($\mathrm{coh}\,(\Delta, \Sigma, n)$) *iff*

- $\Delta = \epsilon$ and $\mathrm{functions}(\Sigma) = \epsilon$, or
- $\Delta = \Delta', f\ (\overline{t_i\ x_i})\ \{\ c\ \}$, and $\Sigma = \Sigma', f\ (\overline{x_i}) : \{\mathcal{P}\}\{\mathcal{Q}\}$, and $\mathrm{coh}\,(\Delta', \Sigma', n)$, and $\Sigma'; \{\overline{x_i}\}\ ; \{\mathcal{P}\}\ c\ \{\mathcal{Q}\}$ is n-valid *wrt.* Δ', or
- $\Delta = \Delta', f\ (\overline{t_i\ x_i})\ \{\ c\ \}$, and $\Sigma = \Sigma', f\ (\overline{x_i}) : \left\{\phi; \lceil P \rceil * p^1(\overline{e_i})\right\}\{\lceil Q \rceil\}$, and $\mathrm{coh}\,(\Delta', \Sigma', n)$, and $\Sigma; \{\overline{x_i}\}\ ; \left\{\lceil P \rceil * p^1(\overline{e_i})\right\}\ c\ \{\lceil Q \rceil\}$ is n'-valid *wrt.* Δ for all $n' < n$.

Theorem 3.3 (Soundness of SSL). *For any n, Δ', if*

(i) $\Sigma'; \Gamma; \{\mathcal{P}\} \rightsquigarrow \{\mathcal{Q}\} | c$ *for a goal named f with formal parameters $\Gamma \triangleq \overline{x_i}$, and*

(ii) Σ' *is such that* $\mathrm{coh}\,(\Delta', \Sigma', n)$*, and*

(iii) *for all $p^0(\overline{e_i}), \phi; P$, such that $\{\mathcal{P}\} = \{\phi; p^0(\overline{e_i}) * P\}$, taking $\mathcal{F} \triangleq f(\overline{x_i}) : \{\phi; p^1(\overline{e_i}) * \lceil P \rceil\}\{\lceil \mathcal{Q} \rceil\}$,*
$\Sigma', \mathcal{F}; \Gamma; \{\mathcal{P}\}\ c\ \{\mathcal{Q}\}$ *is n'-valid for all $n' < n$ wrt. $\Delta \triangleq \Delta', f\ (\overline{t_i\ x_i})\ \{\ c\ \}$,*

then $\Sigma'; \Gamma; \{\mathcal{P}\}\ c\ \{\mathcal{Q}\}$ is n-valid wrt. Δ.

Proof. By the top-level induction on n and by inner induction on the structure of derivation $\Sigma'; \Gamma; \{\mathcal{P}\} \rightsquigarrow \{\mathcal{Q}\} | c$. We refer the reader to Appendix A for the details. $\square$

Algorithme de synthèse basé sur SSL

- Recherche en profondeur dans l'espace des dérivations SSL valides.
- Recherche récursive avec backtracking.

Pour synthétiser un programme, besoin de

- contexte et environnement ;
- préconditions et postconditions ;
- liste de règles possibles ;

Optimisations et extensions

Optimisations :

- Règles inversibles

Optimisations et extensions

Optimisations :

- Règles inversibles
- Recherche multi-phase

Optimisations et extensions

Optimisations :

- Règles inversibles
- Recherche multi-phase
- Rèduction des symétries

Optimisations et extensions

Optimisations :

- Règles inversibles
- Recherche multi-phase
- Rèduction des symétries
- Règles d'échec

Optimisations et extensions

Optimisations :

- Règles inversibles
- Recherche multi-phase
- Rèduction des symétries
- Règles d'échec

Extensions :

- Fonctions auxilliaire

Optimisations et extensions

Optimisations :

- Règles inversibles
- Recherche multi-phase
- Rèduction des symétries
- Règles d'échec

Extensions :

- Fonctions auxilliaire
- Enlèvement de branches

Benchmark

Group	*Description*	*Code*	*Code/Spec*	*Time*	*T-phase*	*T-inv*	*T-fail*	*T-com*	*T-all*	*T-IS*
Integers	swap two	12	0.9x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	min of two[2]	10	0.7x	0.1	0.1	0.1	< 0.1	0.1	0.2	
Linked List	length[1,2]	21	1.2x	0.4	0.9	0.5	0.4	0.6	1.4	29x
	max[1]	27	1.7x	0.6	0.8	0.5	0.4	0.4	0.8	20x
	min[1]	27	1.7x	0.5	0.9	0.5	0.4	0.5	1.2	49x
	singleton[2]	11	0.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	dispose	11	2.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	initialize	13	1.4x	< 0.1	0.1	0.1	< 0.1	0.1	< 0.1	
	copy[3]	35	2.5x	0.2	0.3	0.3	0.1	0.2	-	
	append[3]	19	1.1x	0.2	0.3	0.3	0.2	0.3	0.7	
	delete[3]	44	2.6x	0.7	0.5	0.3	0.2	0.3	0.7	
Sorted list	prepend[1]	11	0.3x	0.2	1.4	83.5	0.1	0.1	-	48x
	insert[1]	58	1.2x	4.8	-	-	-	5.0	-	6x
	insertion sort[1]	28	1.3x	1.1	1.8	1.3	1.2	1.2	74.2	82x
Tree	size	38	2.7x	0.2	0.3	0.2	0.2	0.2	0.3	
	dispose	16	4.0x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	copy	55	3.9x	0.4	49.8	-	0.8	1.4	-	
	flatten w/append	48	4.0x	0.4	0.6	0.5	0.4	0.4	0.6	
	flatten w/acc	35	1.9x	0.6	1.7	0.7	0.5	0.6	-	
BST	insert[1]	58	1.2x	31.9	-	-	-	-	-	11x
	rotate left[1]	15	0.1x	37.7	-	-	-	-	-	0.5x
	rotate right[1]	15	0.1x	17.2	-	-	-	-	-	0.8x

L'algorithme

Algorithm 4.1: synthesize ($\mathcal{G}$: Goal, *rules* : Rule*)

Input: Goal $\mathcal{G} = \langle f, \Sigma, \Gamma, \{\mathcal{P}\}, \{\mathcal{Q}\}\rangle$
Input: List *rules* of available rules to try
Result: Program c, such that

$$\Sigma; f\ (\Gamma)\ \{c\}\ ; \Gamma\ \{\mathcal{P}\}\ c\ \{\mathcal{Q}\}$$

```
function synthesize (G, rules) =
  withRules(rules, G)
function withRules (rs, G) =
  match rs
    case [ ] ⇒ Fail
    case R :: rs′ ⇒
      subderivs = filterComm( R(G) )
      if isEmpty(subderivs) then
        withRules(rs′)
      else
        tryAlts(subderivs, R, rs′, G)
function tryAlts (derivs, R, rs, G) =
  match derivs
    case [ ] ⇒ if isInvert(R) then Fail else withRules(rs, G)
    case ⟨goals, K⟩ :: derivs′ ⇒
      match solveSubgoals(goals, K)
        case Fail ⇒ tryAlts(derivs′, R, rs, G)
        case c ⇒ if c = magic then tryAlts(derivs′, R, rs, G) else c
function solveSubgoals (goals, K) =
  cs := [ ]
  pickRules = λG. phasesEnabled ? nextRules(G) : AllRules
  for G ← goals; c = synthesize(G, pickRules(G)); c ≠ Fail do
    cs := cs ++ [c]
  if |cs| < |goals| then Fail else K(cs)
```