Razonamiento local con abstracción y estructuras compartidas

Renato Cherini

Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Octubre de 2008

Introducción a SL y motivaciones

Resumen

El razonamiento local dado por la Separation Logic es una gran herramienta para la verificación de programas con un manejo complejo de punteros. Sin embargo esta lógica encuentra su límite cuando es necesario especificar diversas estructuras que comparten el heap.

Presentamos una generalización de la Separation Logic que permite especificar precisamente estructuras complejas en el heap, relaciones de sharing entre ellas, y un sistema de prueba composicional asociado.

Separation Logic

- Extiende la lógica de Hoare con *operadores espaciales* ($*$, $-\!\!*$).
- Promueve el *razonamiento local* con la *Regla de Frame*:

$$\frac{\{P\}\, C\, \{Q\}}{\{P * R\}\, C\, \{Q * R\}}$$

$$(s,h) \models P * R \;\Leftrightarrow\; \exists h_1, h_2 \cdot h_1 \uplus h_2 = h \wedge (s,h_1) \models P \wedge (s,h_2) \models R$$

- La conjunción espacial $*$ codifica la ausencia de *aliasing*.
- Ha sido aplicada exitosamente en diversos casos de estudio.
- *"SL makes earlier attempts to prove pointer-mutating programs look ridiculously complicated and ad-hoc"*
  (R. Bornat, C. Calcagno, P. O'Hearn, M. Parkinson)

Limitaciones

- La *Regla de Constancia* ya no es válida:

$$\frac{\{P\}\, C\, \{Q\}}{\{P \wedge R\}\, C\, \{Q \wedge R\}}$$

$$(s,h) \models P \wedge R \;\Leftrightarrow\; (s,h) \models P \wedge (s,h) \models R$$

- La conjunción (clasica) $\wedge$ puede introducir múltiples formas de *aliasing*
- PERO en ciertas circustancias el *aliasing* es inevitable!

Limitaciones

$$\frac{\{x \mapsto X\}\,[x] := Z\,\{x \mapsto Z\}}{\{x \mapsto X \ \wedge \ y \mapsto X\}\,[x] := Z\,\{x \mapsto Z \ \wedge \ y \mapsto X\}}$$

$$(s,h) \models P \wedge R \ \Leftrightarrow \ (s,h) \models P \wedge (s,h) \models R$$

- Es válido? Solo si $x \neq y$ (o $X = Z$).

$$\frac{\{x \mapsto X\}\,[x] := Z\,\{x \mapsto Z\}}{\{x \mapsto X \ * \ y \mapsto X\}\,[x] := Z\,\{x \mapsto Z \ * \ y \mapsto X\}}$$

$$(s,h) \models P * R \ \Leftrightarrow \ \exists h_1, h_2 \cdot h_1 \uplus h_2 = h \wedge (s,h_1) \models P \wedge (s,h_2) \models R$$

Motivaciones: Patrón Iterador

Un iterador es utilizado para acceder y modificar una colección mutable, con las siguientes restricciones sobre los métodos:

$$\begin{array}{rcl} \text{Observación} & \rightarrow & \text{permitido} \\ \text{Modificación NO-ESTRUCTURAL} & \rightarrow & \text{permitido} \\ \text{Modificación ESTRUCTURAL} & \rightarrow & \text{prohibido} \end{array}$$

En SL podemos utilizar una especificación *naive* como la siguiente:

$$(\mathbf{iter}.i_0.c.xs * \mathbf{true}) \wedge (\mathbf{coll}.c.xs * \mathbf{true}) \wedge (\mathbf{iter}.i_1.c.xs * \mathbf{true})$$

Lenguaje de aserciones

Nuevas formas de aserciones

- Extendemos el lenguaje de aserciones de la lógica de Hoare:

$$
\begin{array}{lcl}
(s,h) \models \mathbf{emp} & \Leftrightarrow & dom_h = \emptyset \\
(s,h) \models e \mapsto e' & \Leftrightarrow & h = \{([\![e]\!].s, [\![e']\!].s)\} \\
(s,h) \models P \ \langle * : R \rangle \ Q & \Leftrightarrow & \exists h_1, h_2, h_3 \cdot h_1 \uplus h_2 \uplus h_3 = h \ \wedge \\
 & & (s, h_1 \uplus h_3) \models P \ \wedge (s, h_2 \uplus h_3) \models Q \ \wedge (s, h_3) \models R \\
(s,h) \models Q \ \langle -\!\!* : R \rangle \ P & \Leftrightarrow & \forall h_1 \cdot (\exists h_2 \cdot h_2 \subseteq h \wedge (s, h_1 \uplus h_2) \models Q \ \wedge (s, h_2) \models R) \\
 & & \Rightarrow (s, h \uplus h_1) \models P \\
(s,h) \models Q \ \langle -\!\!\circledast : R \rangle \ P & \Leftrightarrow & \exists h_1, h_2 \cdot h_2 \subseteq h \wedge (s, h_1 \uplus h_2) \models Q \ \wedge (s, h_2) \models R \\
 & & \wedge (s, h \uplus h_1) \models P
\end{array}
$$

- Esto representa una generalización de la Separation Logic:

$$P * Q \equiv P \ \langle * : \mathbf{emp} \rangle \ Q$$
$$Q -\!\!* P \equiv Q \ \langle -\!\!* : \mathbf{emp} \rangle \ P$$

Ejemplos de aserciones

$x \mapsto A$	(x, A)
$x \mapsto A * y \mapsto A$	$(x, A)(y, A)$
$x \mapsto A \wedge y \mapsto A$	$(x, A) \wedge x = y$
$x \mapsto A * x \mapsto B$	false
$x \mapsto A \left\langle * : x \mapsto \_ \right\rangle x \mapsto B$	$(x, A) \wedge A = B$
$(x \mapsto A * z \mapsto C) \left\langle * : z \mapsto C \right\rangle (y \mapsto C * z \mapsto C)$	$(x, A)(y, B)(z, C)$

¿Por qué no usar simplemente?

$$x \mapsto A * z \mapsto C * y \mapsto B$$

Propiedades

- Conmutatividad:

$$P \ \langle * : R \rangle \ Q \equiv Q \ \langle * : R \rangle \ P$$

- Elemento neutro:

$$\begin{array}{rcl} P * \mathbf{emp} & \equiv & P \\ P \ \langle * : R_0 * R_1 \rangle \ Q & \Rightarrow & P \ \langle * : R_0 \rangle \ (R_1 \mathrel{-\!\circledast} Q) \end{array}$$

$$\frac{P \Rightarrow (R_0 * R_1 * \mathbf{true})}{P \ \langle * : R_0 \rangle \ Q \Rightarrow P \ \langle * : R_0 * R_1 \rangle \ (R_1 * Q)}$$

- Monotonía:

$$\frac{P_0 \Rightarrow P_1 \quad Q_0 \Rightarrow Q_1 \quad R_0 \Rightarrow R_1}{P_0 \ \langle * : R_0 \rangle \ Q_0 \Rightarrow P_1 \ \langle * : R_1 \rangle \ Q_1}$$

Propiedades (cont.)

- Adjuntividad:

$$\frac{P_0 \ \langle * : R \rangle \ P_1 \ \Rightarrow \ Q}{P_0 \ \Rightarrow \ (P_1 \ \langle -\!\!* : R \rangle \ Q)} \qquad \frac{P \ \Rightarrow \ (Q_0 \ \langle -\!\!* : R \rangle \ Q_1)}{P \ \langle * : R \rangle \ Q_0 \ \Rightarrow \ Q_1}$$

- (Semi) Distributividad:

$$P \ \langle * : R_0 \vee R_1 \rangle \ Q \equiv (P \ \langle * : R_0 \rangle \ Q) \vee (P \ \langle * : R_1 \rangle \ Q)$$
$$(P_0 \vee P_1) \ \langle * : R \rangle \ Q \equiv (P_0 \ \langle * : R \rangle \ Q) \vee (P_1 \ \langle * : R \rangle \ Q)$$
$$(P_0 \wedge P_1) \ \langle * : R \rangle \ Q \Rightarrow (P_0 \ \langle * : R \rangle \ Q) \wedge (P_1 \ \langle * : R \rangle \ Q)$$
$$(\exists v \cdot P) \ \langle * : R \rangle \ Q \equiv (\exists v \cdot P \ \langle * : R \rangle \ Q) \quad (v \notin FV.Q \cup FV.R)$$
$$(\forall v \cdot P) \ \langle * : R \rangle \ Q \Rightarrow (\forall v \cdot P \ \langle * : R \rangle \ Q) \quad (v \notin FV.Q \cup FV.R)$$

Propiedades (cont.)

Para ciertas *clases semánticas* de aserciones tenemos:

- Distributividad completa:

$$(P_0 \wedge P_1)\ \langle * : R \rangle\ Q \Leftarrow (P_0\ \langle * : R \rangle\ Q) \wedge (P_1\ \langle * : R \rangle\ Q)$$
$$(\forall v\ \cdot\ P)\ \langle * : R \rangle\ Q \Leftarrow (\forall v\ \cdot\ P\ \langle * : R \rangle\ Q) \quad (v \notin FV.Q \cup FV.R)$$

- Regla de Intercambio, subsume la asociatividad:

$$\frac{P_1 \Rightarrow (R_1 * \mathbf{true})}{(P_0\ \langle * : R_0 \rangle\ P_1)\ \langle * : R_1 \rangle\ P_2 \Rightarrow P_0\ \langle * : R_0 \rangle\ (P_1\ \langle * : R_1 \rangle\ P_2)}$$

Sistema de prueba

Lenguaje de programación

Extendemos el lenguaje de programación de la SL para soportar procedimientos y módulos:

$$
\begin{aligned}
C ::= {} & x := e \mid \textbf{skip} \mid C; C \mid \textbf{if } b \textbf{ then } C \textbf{ else } C \textbf{ fi} \mid \textbf{while } b \textbf{ do } C \textbf{ od} \\
& \mid x := \textbf{cons}(e_1, \ldots, e_n) && \text{(Asignación)} \\
& \mid x := [e] && \text{(Observación)} \\
& \mid [e] := e' && \text{(Modificación)} \\
& \mid \textbf{dispose}(e) && \text{(Destrucción)} \\
& \mid \textbf{newvar}(v); C && \text{(Variable local)} \\
& \mid \textbf{return}(e) && \text{(Retorno)} \\
& \mid \textbf{let } k_1(\overline{x_1}) = C_1 \ldots, k_n(\overline{x_n}) = C_n \textbf{ in } C && \text{(Definición de módulo)} \\
& \mid v := k(\overline{e}) && \text{(Llamada a procedimiento)}
\end{aligned}
$$

Especificaciones

Extendemos las triplas de Hoare con un contexto de *predicados de abstracción* Λ, especificaciones de procedimientos Γ y *especificaciones estáticas* Σ:

$$spec ::= \Lambda; \Gamma; \Sigma \vdash \{P\}\ C\ \{Q\}$$

$$\Lambda ::= \epsilon \mid \alpha.\overline{x} \doteq P, \Lambda$$
$$\Gamma ::= \epsilon \mid \{P\}k(\overline{x})\{Q\}, \Gamma$$
$$\Sigma ::= \epsilon \mid P \Rightarrow Q, \Sigma$$

Abstracción y ocultamiento de información

- Los *predicados de abstracción* modelan TADs. Poseen:
  - Un nombre
  - Una definición (implementación)
  - Un *scope*
- Las *especificaciones estáticas* son pasos de deducción (abstracta) válidos.
- Λ y Σ son hipótesis para la satisfacción de una formula:

$$\alpha.\overline{x} \doteq P, \Lambda; \Sigma \models \alpha.\overline{e} \equiv P_{/\overline{x} \leftarrow \overline{e}}$$
$$\Lambda; P \Rightarrow Q, \Sigma \models P_{/\overline{x} \leftarrow \overline{e}} \Rightarrow Q_{/\overline{x} \leftarrow \overline{e}}$$

Reglas para los comandos

- Asignación:

$$\Lambda, \Gamma, \Sigma \vdash \{v = v' \wedge \mathbf{emp}\}\, v := \mathbf{cons}(\overline{e})\, \{v \mapsto \overline{e}_{/v \leftarrow v'}\}$$

  cuando v y v' son distintas

- Observación:

$$\Lambda, \Gamma, \Sigma \vdash \{v = v' \wedge e \mapsto e'\}\, v := [e]\, \{v = e'_{/v \leftarrow v'} \wedge e_{/v \leftarrow v'} \mapsto v\}$$

  cuando v y v' son distintas

- Modificación:

$$\Lambda, \Gamma, \Sigma \vdash \{e \mapsto \_\}\, [e] := e'\, \{e \mapsto e'\}$$

- Destrucción:

$$\Lambda, \Gamma, \Sigma \vdash \{e \mapsto \_\}\, \mathbf{dispose}(e)\, \{\mathbf{emp}\}$$

Reglas para los comandos (cont.)

- Variable Local:

$$\frac{\Lambda, \Sigma, \Gamma \vdash \{P_{/x \leftarrow y} \wedge x = \mathbf{nil}\}\, C\, \{Q_{/x \leftarrow y}\}}{\Lambda, \Sigma, \Gamma \vdash \{P\}\, \mathbf{newvar}(x); C\, \{Q\}}$$

cuando $y \notin FV.P \cup FV.C \cup FV.Q$

- Retorno:

$$\Lambda; \Gamma; \Sigma \vdash \{P_{/x \leftarrow ret}\}\, \mathbf{return}(x)\, \{P\}$$

- Llamada a procedimiento:

$$\frac{\Lambda, \Sigma, \Gamma \vdash \{P\}\, k(\overline{x})\, \{Q\} \in \Gamma}{\Lambda; \Gamma; \Sigma \vdash \{P_{/\overline{x} \leftarrow \overline{y}}\}\, v = k(\overline{y})\, \{Q_{/\overline{x}, ret \leftarrow \overline{y}, v}\}}$$

Reglas para los comandos (cont.)

- Definición de módulo:

$$\frac{\begin{array}{c}\Lambda, \Lambda'; \Sigma \models \Sigma' \\ \Lambda, \Lambda'; \Gamma; \Sigma, \Sigma' \vdash \{P_1\}\, C_1\, \{Q_1\} \\ \vdots \\ \Lambda, \Lambda'; \Gamma; \Sigma, \Sigma' \vdash \{P_n\}\, C_n\, \{Q_n\} \\ \Lambda, \Gamma, \{P_1\}k_1(\overline{x_1})\{Q_1\}, \ldots, \{P_n\}k_n(\overline{x_n})\{Q_n\}; \Sigma, \Sigma' \vdash \{P\}C\{Q\}\end{array}}{\Lambda; \Gamma; \Sigma \vdash \{P\}\textbf{let } k_1(\overline{x_1}) = C_1, \ldots, k_n(\overline{x_n}) = C_n \textbf{ in } C\{Q\}}$$

cuando
- P,Q,Γ, Λ y Σ no contienen nombres de predicados de $dom(\Lambda')$,
- $dom(\Lambda')$ and $dom(\Lambda)$ son disjuntos, y
- los procedimientos sólo modifican variables locales.

Reglas estructurales

- Regla de Consecuencia:

$$\frac{\Lambda, \Sigma \models P' \Rightarrow P \quad \vdash \{P\}\, C\, \{Q\} \quad \Lambda, \Sigma \models Q \Rightarrow Q'}{\vdash \{P'\}\, C\, \{Q'\}}$$

- Regla de Frame General:

$$\frac{\begin{array}{c}\Lambda, \Sigma, \Gamma \vdash \{P\}\, C\, \{Q\} \\ \Lambda, \Sigma \models (R \mathrel{-\!\!\circledast} I) * R' \Rightarrow I' \\ \Lambda, \Sigma \models Q \Rightarrow (R' * \mathbf{true})\end{array}}{\Lambda, \Sigma, \Gamma \vdash \{P \ \langle * : R \rangle \ I\}\, C\, \{Q \ \langle * : R' \rangle \ I'\}}$$

cuando C no modifica las variables libres de R e I.

Especificación de Iterador

Λ:

$$\begin{aligned}
\mathbf{coll}.c.[\,].[\,] &\doteq c = \mathbf{nil} \wedge \mathbf{emp} \\
\mathbf{coll}.c.(l \triangleright ls).(x \triangleright xs) &\doteq c \mapsto x, l * \mathbf{coll}.l.ls.xs \\
\mathbf{iter}.i.n.c.ls.xs &\doteq \exists k \cdot i \mapsto c, k \ldots * (\mathbf{coll}.c.ls.xs \wedge (n < \#xs \Rightarrow k = ps.n))
\end{aligned}$$

Γ:

$$\begin{array}{rcl}
\{\mathbf{emp}\} & new_coll() & \{\mathbf{coll}.ret.[\,].[\,]\} \\
\{\mathbf{coll}.c.ls.xs\} & add(c,x) & \{(\exists ls \cdot \mathbf{coll}.c.ls.(x \triangleright xs))\} \\
\{\mathbf{coll}.c.ls.(x \triangleright xs)\} & del(c) & \{(\exists ls \cdot \mathbf{coll}.c.ls.xs)\} \\
\{\mathbf{coll}.c.ls.xs\} & size(c) & \{\mathbf{coll}.c.ls.xs \wedge ret = \#xs\} \\
\{\mathbf{coll}.c.ls.xs\} & new_iter(c) & \{\mathbf{iter}.ret,0.c.ls.xs\} \\
\{\mathbf{iter}.i.n.c.ls.xs \wedge n < \#xs - 1\} & next(i) & \{\mathbf{iter}.i.(n+1).c.ls.xs \wedge ret = xs.n\} \\
\{\mathbf{iter}.i.n.c.ls.xs \wedge n < \#xs\} & set(i,x) & \{\mathbf{iter}.i.n.c.ls.(xs[x/n])\} \\
\{\mathbf{iter}.i.n.c.ls.xs\} & index(i) & \{\mathbf{iter}.i.n.c.ls.xs \wedge ret = n\}
\end{array}$$

Σ:

$$\begin{aligned}
&\varsigma_1 : \mathbf{iter}.i.n.c.ls.xs \Rightarrow \mathbf{coll}.c.ls.xs * \mathbf{true} \\
&\varsigma_2 : (\mathbf{coll}.c.ls.xs \multimap\!\circledast \mathbf{iter}.i.n.c.ls.xs) * \mathbf{coll}.c.ls.xs' \Rightarrow \mathbf{iter}.i.n.c.ls.xs' \\
&\varsigma_3 : (\mathbf{coll}.c.ls.xs \multimap\!\circledast \mathbf{coll}.c.ls.xs) * \mathbf{coll}.c.ls.xs' \Rightarrow \mathbf{coll}.c.ls.xs'
\end{aligned}$$

Código cliente de Iterador

$$\{\, \mathbf{coll}.c.ls.xs \ \langle * : \ \mathbf{coll}.c.ls.xs \rangle \ \mathbf{iter}.i_1.0.c.ls.xs \,\}$$

GFR:
$$\left|\begin{array}{l} \{\, \mathbf{coll}.c.ls.xs \,\} \\ \quad i_0 := new\_iter(c); \\ \{\, \mathbf{iter}.i_0.0.c.ls.xs \,\} \\ \quad next(i_0); \\ \{\, \mathbf{iter}.i_0.1.c.ls.xs \,\} \end{array}\right.$$

$$\{\, \mathbf{iter}.i_0.1.c.ls.xs \ \langle * : \ \mathbf{coll}.c.ls.xs \rangle \ \mathbf{iter}.i_1.0.c.ls.xs \,\}$$

GFR:
$$\left|\begin{array}{l} \{\, \mathbf{iter}.i_1.0.c.ls.xs \,\} \\ \quad next(i_1); \\ \quad x := next(i_1); \\ \{\, \mathbf{iter}.i_1.2.c.ls.xs \ \wedge \ x = xs.1 \,\} \\ \quad set(i_1, x); \\ \{\, \mathbf{iter}.i_1.2.c.ls.(xs[xs.1/2]) \,\} \end{array}\right.$$

$$\{\, \mathbf{iter}.i_0.1.c.ls.(xs[xs.1/2]) \ \langle * : \ \mathbf{coll}.c.ls.(xs[xs.1/2]) \rangle \ \mathbf{iter}.i_1.2.c.ls.(xs[xs.1/2]) \,\}$$

GFR:
$$\left|\begin{array}{l} \{\, \mathbf{iter}.i_0.1.c.ls.(xs[xs.1/2]) \,\} \\ \quad y := next(i_0); \\ \{\, \mathbf{iter}.i_0.2.c.ls.(xs[xs.1/2]) \ \wedge \ y = xs.0 \,\} \end{array}\right.$$

$$\{\, (\mathbf{iter}.i_0.2.c.ls.xs[xs.1/2] \ \wedge \ y = xs.0) \ \langle * : \ \mathbf{coll}.c.ls.(xs[xs.1/2]) \rangle \ \mathbf{iter}.i_1.2.c.ls.(xs[xs.1/2]) \,\}$$

Código cliente de Iterador

$\{\, \mathbf{emp} \,\}$
$\quad c := new\_coll();$
$\{\, \mathbf{coll}.c.[\,].[\,] \,\}$
$\quad add(c, x_0);$
$\quad add(c, x_1);$
$\{\, (\exists ls \;\cdot\; \mathbf{coll}.c.ls.[x_1, x_0]) \,\}$
$\Rightarrow \langle$ Neutral element $\rangle$
$\{\, (\exists ls \;\cdot\; \mathbf{coll}.c.ls.[x_1, x_0] \;\langle * : \mathbf{coll}.c.ls.[x_1, x_0] \rangle\; \mathbf{coll}.c.ls.[x_1, x_0]) \,\}$

$\{\, \mathbf{coll}.c.ls.[x_1, x_0] \;\langle * : \mathbf{coll}.c.ls.[x_1, x_0] \rangle\; \mathbf{coll}.c.ls.[x_1, x_0] \,\}$

GFR:
$\{\, \mathbf{coll}.c.ls.[x_1, x_0] \,\}$
$\quad i := new\_iter(c);$
$\{\, \mathbf{iter}.i,0.c.ls.[x_1, x_0] \,\}$

VAR:
$\{\, (\mathbf{iter}.i,0.c.ls.[x_1, x_0]) \;\langle * : \mathbf{coll}.c.ls.[x_1, x_0] \rangle\; \mathbf{coll}.c.ls.[x_1, x_0] \,\}$

GFR:
$\{\, \mathbf{coll}.c.ls.[x_1, x_0] \,\}$
$\quad add(c, x_2);$
$\{\, (\exists ls \;\cdot\; \mathbf{coll}.c.ls.[x_2, x_1, x_0]) \,\}$

$\{\, \mathbf{true} * (\exists ls \;\cdot\; \mathbf{coll}.c.ls.[x_2, x_1, x_0]) \,\}$

$\{\, (\exists ls \;\cdot\; \mathbf{true} * \mathbf{coll}.c.ls.[x_2, x_1, x_0]) \,\}$

Conclusiones y trabajo futuro

Sumario

- Extensión del lenguaje de aserciones para especificar relaciones de *sharing*.
- *Predicados de abstraccion* para modelar ocultamiento de información.
- *Especificaciones estáticas* para razonar de manera abstracta.
- Regla de definición de módulo, regla de consecuencia y regla de *frame* para juntar las piezas.

Trabajo Futuro

- Modelización en Coq (consistencia y completitud).
- Extensión a un lenguaje de programación *Java-like* (*subclassing*, *dynamic dispatch*, etc).
- Extensión a un entorno concurrente basado Rely/Guarantee + Separation Logic.

¿Preguntas?