CPL seminar

2011-07-18

(with tiny revisions)

Georg Neis

Today:

Mike Dodds, Xinyu Feng, Matthew J. Parkinson, Viktor Vafeiadis: Deny-Guarantee Reasoning.

ESOP 2009: 363-377

Overview

- Generalization of Rely-Guarantee
- Program logic for dynamic concurrency
  - Fork/join; interference changes over time
- Separating conjunction splits interference (with the help of fractional permissions)
- Soundness wrt highly instrumented semantics (technical appendix contains erasure details)

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Raw language semantics

- Local semantics (single thread):

  $$(C, \sigma) \rightsquigarrow_m (C', \sigma')$$

- Global semantics (collection of threads):

  $$(\sigma, \delta) \Longmapsto (\sigma', \delta')$$

- Simplifying restriction: no memory allocation, no local variables

Raw language semantics

$$\frac{[\![E]\!]_\sigma = n}{(x := E, \sigma) \rightsquigarrow_m (\mathbf{skip}, \sigma[x \mapsto n])} \qquad \frac{}{(\mathbf{skip}; C, \sigma) \rightsquigarrow_m (C, \sigma)} \qquad \frac{(C, \sigma) \rightsquigarrow_m (C', \sigma')}{(C; C', \sigma) \rightsquigarrow_m (C'; C', \sigma')}$$

$$\frac{}{(x := \mathbf{fork}\ C, \sigma) \overset{\mathbf{fork}\ (t,C)}{\rightsquigarrow_m} (\mathbf{skip}, \sigma[x \mapsto t])} \qquad \frac{[\![E]\!]_\sigma = t}{(\mathbf{join}\ E, \sigma) \overset{\mathbf{join}\ t}{\rightsquigarrow_m} (\mathbf{skip}, \sigma)}$$

$$\frac{\widehat{\delta}(t) = C \quad (C, \sigma) \rightsquigarrow_m (C', \sigma')}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'])} \qquad \frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{fork}\ (t_2,C_2)}{\rightsquigarrow_m} (C', \sigma') \quad t_2 \notin dom(\widehat{\delta})}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'] \uplus [t_2 \mapsto C_2])}$$

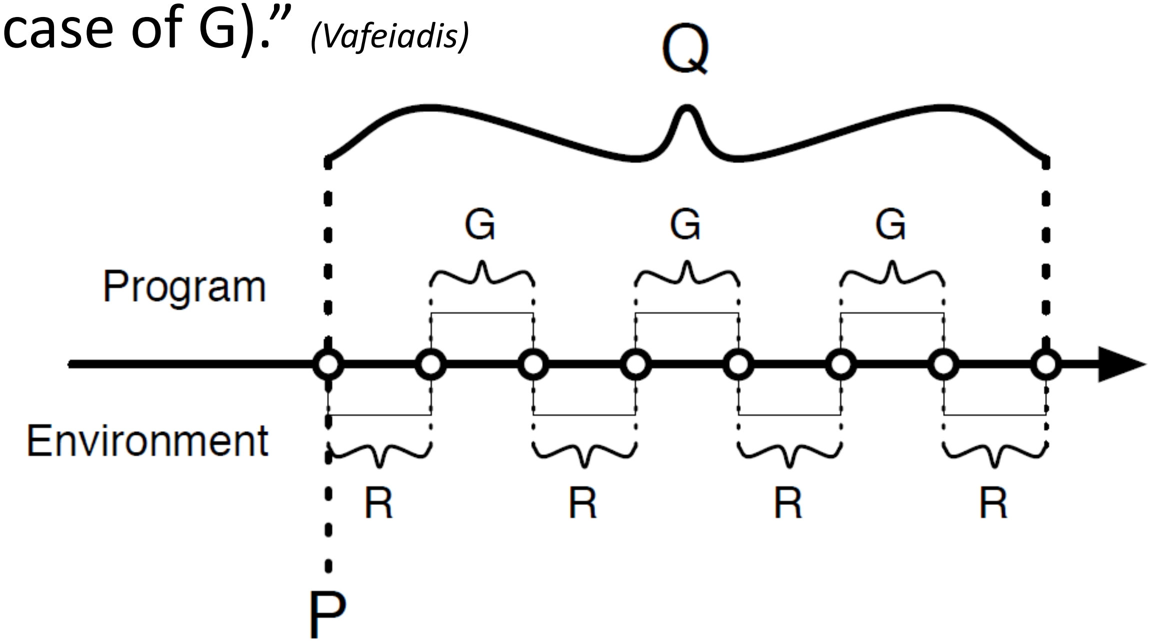
$$\frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{join}\ t_2}{\rightsquigarrow_m} (C', \sigma') \quad \widehat{\delta}(t_2) = \mathbf{skip}}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'] \setminus t_2\})} \qquad \frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{join}\ t_2}{\rightsquigarrow_m} (C', \sigma') \quad t_2 \notin dom(\widehat{\delta})}{(\sigma, \widehat{\delta}) \Longmapsto_m \mathbf{abort}}$$

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Recall: Rely-guarantee conditions

- Interference becomes part of specification, in the form of two binary relations: $\{P, \underline{R}\} \vdash S\ \textbf{sat}\ (\underline{G}, Q)$
- "R and G summarise the properties of the individual atomic actions invoked by the environment (in the case of R) and the thread itself (in the case of G)." *(Vafeiadis)*



From R-G to D-G

- Recall Rely-Guarantee:

$$\frac{R_1, G_1 \vdash \{P_1\}\, C_1\, \{Q_1\} \quad G_1 \subseteq R_2 \quad R_2, G_2 \vdash \{P_2\}\, C_2\, \{Q_2\} \quad G_2 \subseteq R_1}{R_1 \cap R_2, G_1 \cup G_2 \vdash \{P_1 \wedge P_2\}\, C_1 \,\|\, C_2\, \{Q_1 \wedge Q_2\}}$$

- Doesn’t make sense for dynamic concurrency:

  Interference before a fork is not the same as after a fork

From R-G to D-G

- What about specs of the following form? $\{(\mathrm{R}, \mathrm{G}), \mathrm{P}\}\ \mathrm{C}\ \{(\mathrm{R}', \mathrm{G}'), \mathrm{P}'\}$

- Attempt to adapt R-G rule: $\dfrac{\{(R_1,G_1),P\}\ \mathrm{C}\ \{\dots\} \quad G_1 \subseteq R_2 \wedge G_2 \subseteq R_1}{\{(R,G),P\}\ \mathrm{fork\ C}\ \{(R_2,G_2),\dots\}}$

- Rewrite as $\dfrac{\{(R_1,G_1),P\}\ \mathrm{C}\ \{\dots\}}{\{(R_1,G_1) * (R_2,G_2),\ P\}\ \mathrm{fork\ C}\ \{(R_2,G_2),\dots\}}$ via separation between R,G:

$$(R_1, G_1) * (R_2, G_2) = (R_1 \cap R_2, G_1 \cup G_2) \quad \text{if } G_1 \subseteq R_2 \wedge G_2 \subseteq R_1$$

- Doesn't work: no cancellativity

$$A * B_1 = A * B_2 \ \Rightarrow\ B_1 = B_2$$
$$(R, G) * (R_1, G_1) = (R, G) * (R_2, G_2) \ \Rightarrow\ (R_1, G_1) = (R_2, G_2)$$

(Not per se unsound, but traditional approach to proving soundness not applicable)

From R-G to D-G

- But idea is right! Want simple rules:

$$\frac{\{P_1\}\ C\ \{P_2\} \quad \ldots}{\{P * P_1\}\ x := \mathbf{fork}\ C\ \{P * \mathsf{Thread}(x, P_2)\}}\ (\text{FORK})$$

$$\frac{\ldots}{\{P * \mathsf{Thread}(E, P')\}\ \mathbf{join}\ E\ \{P * P'\}}\ (\text{JOIN})$$

- Let’s do this, and throw in fractional permissions, too.

Permissions

Can look at the store

$$
\begin{array}{rcll}
\sigma \in & \text{States} & \stackrel{\textbf{def}}{=} & \text{Vars} \longrightarrow \text{Vals} \\
a \in & \text{Actions} & \stackrel{\textbf{def}}{=} & \{\sigma[x \mapsto n], \sigma[x \mapsto n'] \mid \sigma \in \text{States} \ \wedge \ n \neq n'\} \\
f \in & \text{FractionDG} & \stackrel{\textbf{def}}{=} & \{(\text{deny}, \pi) \mid \pi \in (0,1)\} \cup \{(\text{guar}, \pi) \mid \pi \in (0,1)\} \cup \{0, 1\} \\
pr \in & \text{PermDG} & \stackrel{\textbf{def}}{=} & \text{Actions} \longrightarrow \text{FractionDG}
\end{array}
$$

$$
\begin{tikzcd}
 & \overset{(me, \neg env)}{1} \arrow[dl] \arrow[dr] & \\
\underset{(me, env)}{guar} \arrow[dr] & & \underset{(\neg me, \neg env)}{deny} \arrow[dl] \\
 & \underset{(\neg me, env)}{0} &
\end{tikzcd}
$$

Permissions

$$
\begin{tikzcd}
& \overset{(me,\neg env)}{1} \arrow[dl] \arrow[dr] & \\
\underset{(me,env)}{guar} \arrow[dr] & & \underset{(\neg me,\neg env)}{deny} \arrow[dl] \\
& \underset{(\neg me,env)}{0} &
\end{tikzcd}
$$

$$
\begin{aligned}
0 \oplus x \stackrel{\textbf{def}}{=} x \oplus 0 &\stackrel{\textbf{def}}{=} x \\
(\text{deny},\pi) \oplus (\text{deny},\pi') &\stackrel{\textbf{def}}{=} \text{if } \pi + \pi' < 1 \text{ then } (\text{deny}, \pi + \pi') \\
&\qquad \text{else if } \pi + \pi' = 1 \text{ then } 1 \text{ else undef} \\
(\text{guar},\pi) \oplus (\text{guar},\pi') &\stackrel{\textbf{def}}{=} \text{if } \pi + \pi' < 1 \text{ then } (\text{guar}, \pi + \pi') \\
&\qquad \text{else if } \pi + \pi' = 1 \text{ then } 1 \text{ else undef} \\
1 \oplus x \stackrel{\textbf{def}}{=} x \oplus 1 &\stackrel{\textbf{def}}{=} \text{if } x = 0 \text{ then } 1 \text{ else undef}
\end{aligned}
$$

Addition is commutative, associative, cancellative, and has 0 as a unit element. Lifting addition pointwise to $pr \in \mathrm{PermDG}$, can define a separation logic.

Permission examples

- Notation:

$$x : A \rightsquigarrow B \stackrel{\mathbf{def}}{=} \{(\sigma[x \mapsto v], \sigma[x \mapsto v']) \mid \sigma \in \mathsf{State} \wedge v \in A \wedge v' \in B \wedge v \neq v'\}$$

$$[X]_f \stackrel{\mathbf{def}}{=} \lambda a. \begin{cases} f & \text{if } a \in X \\ 0 & \text{otherwise} \end{cases}$$

- Example: $[\mathbf{x} : \mathbb{Z} \rightsquigarrow \{1,2,3\}]_1$
- Splitting property:

$$[x : A \rightsquigarrow B \uplus B']_f \iff [x : A \rightsquigarrow B]_f * [x : A \rightsquigarrow B']_f$$

$$[x : A \rightsquigarrow B]_{f \oplus f'} \iff [x : A \rightsquigarrow B]_f * [x : A \rightsquigarrow B]_{f'}$$

- Example:

$$[\mathbf{x} : \mathbb{Z} \rightsquigarrow \{1,2,3\}]_1 \iff [\mathbf{x} : \mathbb{Z} \rightsquigarrow 1]_1 * [\mathbf{x} : \mathbb{Z} \rightsquigarrow 2]_1 * [\mathbf{x} : \mathbb{Z} \rightsquigarrow 3]_1$$

Permission examples

Precise
wrt. what?

- Another splitting property:

  If P precise and satisfiable, then:

$$(P \mathrel{-\!\!*} \mathsf{full}) * P \iff \mathsf{full}$$

- Example:

$$\mathsf{full} \iff ([\mathrm{x} : \mathbb{Z} \leadsto \{1,2,3\}]_1 \mathrel{-\!\!*} \mathsf{full}) * [\mathrm{x} : \mathbb{Z} \leadsto \{1,2,3\}]_1$$

Extracting R-G

$$
\begin{tikzcd}
 & \overset{(me, \neg env)}{1} \arrow[dl] \arrow[dr] & \\
\underset{(me, env)}{guar} \arrow[dr] & & \underset{(\neg me, \neg env)}{deny} \arrow[dl] \\
 & \underset{(\neg me, env)}{0} &
\end{tikzcd}
$$

$$
[\![\_]\!] \quad \in \ \mathsf{PermDG} \rightarrow \overset{pr.R}{\mathcal{P}(\mathsf{Actions})} \times \overset{pr.G}{\mathcal{P}(\mathsf{Actions})}
$$

$$
\begin{aligned}
[\![pr]\!] \overset{\mathbf{def}}{=} (&\{a \mid pr(a) = (\mathsf{guar}, \_) \vee pr(a) = 0\}, \\
&\{a \mid pr(a) = (\mathsf{guar}, \_) \vee pr(a) = 1\})
\end{aligned}
$$

Assertions

$$P, Q ::= B \mid pr \mid \cancel{\mathsf{full}} \mid \cancel{\mathsf{false}} \mid \mathsf{Thread}(E, P) \mid P \Rightarrow Q \mid P * Q \mid P \mathrel{-\!\!*} Q \mid \exists x. P$$

$$\sigma, pr, \gamma \models B \iff ([\![B]\!]_\sigma = \mathsf{tt}) \wedge (\forall a.\, pr(a) = 0) \wedge (\gamma = \emptyset)$$

$$\sigma, pr, \gamma \models pr' \iff (\gamma = \emptyset) \wedge (pr = pr')$$

$$\sigma, pr, \gamma \models \mathsf{full} \iff (\gamma = \emptyset) \wedge (\forall a.\, pr(a) = 1) \qquad pr = \lambda a. 1$$

$$\sigma, pr, \gamma \models \mathsf{Thread}(E, P) \iff \gamma = [[\![E]\!]_\sigma \mapsto P]$$

$$\sigma, pr, \gamma \models P_1 * P_2 \iff \exists pr_1, pr_2, \gamma_1, \gamma_2.\ pr = pr_1 \oplus pr_2 \wedge \gamma = \gamma_1 \uplus \gamma_2 \wedge (\sigma, pr_1, \gamma_1 \models P_1) \wedge (\sigma, pr_2, \gamma_2 \models P_2)$$

where $\uplus$ means the union of disjoint sets.

$$\sigma, pr, \gamma \models P_1 \mathrel{-\!\!*} P_2 \iff \forall pr_1, pr_2, \gamma_1, \gamma_2.\ pr_2 = pr \oplus pr_1 \wedge \gamma_2 = \gamma \uplus \gamma_1 \wedge (\sigma, pr_1, \gamma_1 \models P_1) \text{ implies } (\sigma, pr_2, \gamma_2 \models P_2)$$

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Recall: Raw language semantics

- Local semantics (single thread):

$$(C,\sigma) \rightsquigarrow_m (C',\sigma')$$

- Global semantics (collection of threads):

$$(\sigma,\delta) \Longmapsto (\sigma',\delta')$$

- Simplifying restriction: no memory allocation, no local variables

Instrumented language semantics

- Local semantics (single thread):

$$\cancel{(C,\sigma)} \rightsquigarrow_m \cancel{(C',\sigma')} \qquad \boxed{(C,\sigma,pr,\gamma)}$$

- Global semantics (collection of threads):

$$(\sigma,\delta) \Longrightarrow (\sigma',\delta')$$

- Simplifying restriction: no memory allocation, no local variables

Recall: Raw language semantics

$$\frac{[\![E]\!]_\sigma = n}{(x := E, \sigma) \rightsquigarrow_m (\mathbf{skip}, \sigma[x \mapsto n])} \qquad \frac{}{(\mathbf{skip}; C, \sigma) \rightsquigarrow_m (C, \sigma)} \qquad \frac{(C, \sigma) \rightsquigarrow_m (C', \sigma')}{(C; C', \sigma) \rightsquigarrow_m (C'; C', \sigma')}$$

$$\frac{}{(x := \mathbf{fork}\ C, \sigma) \overset{\mathbf{fork}\ (t,C)}{\rightsquigarrow_m} (\mathbf{skip}, \sigma[x \mapsto t])} \qquad \frac{[\![E]\!]_\sigma = t}{(\mathbf{join}\ E, \sigma) \overset{\mathbf{join}\ t}{\rightsquigarrow_m} (\mathbf{skip}, \sigma)}$$

$$\frac{\widehat{\delta}(t) = C \quad (C, \sigma) \rightsquigarrow_m (C', \sigma')}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'])} \qquad \frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{fork}\ (t_2,C_2)}{\rightsquigarrow_m} (C', \sigma') \quad t_2 \notin dom(\widehat{\delta})}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'] \uplus [t_2 \mapsto C_2])}$$

$$\frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{join}\ t_2}{\rightsquigarrow_m} (C', \sigma') \quad \widehat{\delta}(t_2) = \mathbf{skip}}{(\sigma, \widehat{\delta}) \Longmapsto_m (\sigma', \widehat{\delta}[t \mapsto C'] \setminus t_2\})} \qquad \frac{\widehat{\delta}(t) = C \quad (C, \sigma) \overset{\mathbf{join}\ t_2}{\rightsquigarrow_m} (C', \sigma') \quad t_2 \notin dom(\widehat{\delta})}{(\sigma, \widehat{\delta}) \Longmapsto_m \mathbf{abort}}$$

Instrumented language semantics

$$\frac{[\![E]\!]_\sigma = n \quad (\sigma, \sigma[x \mapsto n]) \in pr.G}{(x := E, \sigma, pr, \gamma) \rightsquigarrow (\mathbf{skip}, \sigma[x \mapsto n], pr, \gamma)} \qquad \frac{[\![E]\!]_\sigma = n \quad (\sigma, \sigma[x \mapsto n]) \notin pr.G}{(x := E, \sigma, pr, \gamma) \rightsquigarrow \mathbf{abort}}$$

$$\frac{[\![E]\!]_\sigma = t \quad \gamma(t) = Q \quad \sigma, pr', \gamma' \models Q}{(\mathbf{join}\ E, \sigma, pr, \gamma) \overset{\mathbf{join}\ (t, pr', \gamma')}{\rightsquigarrow} (\mathbf{skip}, \sigma, pr \oplus pr', (\gamma \setminus t) \uplus \gamma')} \qquad \frac{[\![E]\!]_\sigma = t \quad t \notin dom(\gamma)}{(\mathbf{join}\ E, \sigma, pr, \gamma) \rightsquigarrow \mathbf{abort}}$$

$$\frac{t \notin dom(\gamma) \quad \sigma, pr', \gamma' \models P \quad pr = pr' \oplus pr'' \quad \gamma = \gamma' \uplus \gamma'' \quad (\sigma, \sigma[x \mapsto t]) \in pr.G}{(x := \mathbf{fork}_{[P,Q]}\ C, \sigma, pr, \gamma) \overset{\mathbf{fork}\ (t, C, pr', \gamma')}{\rightsquigarrow} (\mathbf{skip}, \sigma[x \mapsto t], pr'', \gamma''[t \mapsto Q])}$$

$$\frac{\sigma, pr, \gamma \not\models P * \mathsf{true}}{(x := \mathbf{fork}_{[P,Q]}\ C, \sigma, pr, \gamma) \rightsquigarrow \mathbf{abort}} \qquad \frac{(\sigma, \sigma[x \mapsto t]) \notin pr.G}{(x := \mathbf{fork}_{[P,Q]}\ C, \sigma, pr, \gamma) \rightsquigarrow \mathbf{abort}}$$

Only the father can join his children.

Instrumented language semantics

Rules for interference:

Why necessary?

$$\frac{(\sigma,\sigma') \in pr.R}{(C,\sigma,pr,\gamma) \overset{r}{\rightsquigarrow} (C,\sigma',pr,\gamma)}$$

$$\frac{\forall (t \mapsto C, pr, \gamma) \in \delta.\ \ (\sigma,\sigma') \in pr.R}{(\sigma,\delta) \overset{r}{\Longmapsto} (\sigma',\delta)}$$

Instrumented language semantics

$$\frac{(C, \sigma, pr, \gamma) \rightsquigarrow (C', \sigma', pr', \gamma') \quad (\sigma, \delta) \overset{r}{\Longmapsto} (\sigma', \cancel{\delta'})}{(\sigma, [t \mapsto C, pr, \gamma] \uplus \delta) \Longmapsto (\sigma', [t \mapsto C', pr', \gamma'] \uplus \delta')}$$

$$\frac{(C, \sigma, pr, \gamma) \overset{\mathbf{fork}\,(t_2, C_2, pr_2, \gamma_2)}{\rightsquigarrow} (C', \sigma', pr', \gamma') \quad (\sigma, \delta) \overset{r}{\Longmapsto} (\sigma', \cancel{\delta'})}{(\sigma, [t_1 \mapsto C, pr, \gamma] \uplus \delta) \Longmapsto (\sigma', [t \mapsto C', pr', \gamma'] \uplus [t_2 \mapsto C_2, pr_2, \gamma_2] \uplus \delta')}$$

$$\frac{(C, \sigma, pr, \gamma) \overset{\mathbf{join}\,(t_2, pr_2, \gamma_2)}{\rightsquigarrow} (C', \sigma', pr', \gamma') \quad (\sigma, \delta) \overset{r}{\Longmapsto} (\sigma', \cancel{\delta'})}{(\sigma, [t_1 \mapsto C, pr, \gamma] \uplus [t_2 \mapsto \mathbf{skip}, pr_2, \gamma_2] \uplus \delta) \Longmapsto (\sigma', [t \mapsto C', pr', \gamma'] \uplus \delta')}$$

Instrumented language semantics

$$\frac{(C,\sigma,pr,\gamma) \leadsto \mathbf{abort}}{(\sigma,[t \mapsto C,pr,\gamma] \uplus \delta) \Longmapsto \mathbf{abort}} \qquad \frac{(C,\sigma,pr,\gamma) \overset{-}{\leadsto} (C,\sigma',pr',\gamma') \quad \neg(\exists \delta'.\,(\sigma,\delta) \overset{r}{\Longmapsto} (\sigma',\delta'))}{(\sigma,[t \mapsto C,pr,\gamma] \uplus \delta) \Longmapsto \mathbf{abort}}$$

$$\frac{(C,\sigma,pr,\gamma) \overset{\mathbf{join}\,(t_2,pr_3,\gamma_3)}{\leadsto} (C',\sigma',pr',\gamma') \quad \neg((C,\sigma,pr,\gamma) \overset{\mathbf{join}\,(t_2,pr_2,\gamma_2)}{\leadsto} (C',\sigma',pr',\gamma') \quad)}{(\sigma,[t_1 \mapsto C,pr,\gamma] \uplus [t_2 \mapsto \mathbf{skip},pr_2,\gamma_2] \uplus \delta) \Longmapsto \mathbf{abort}}$$

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

The Rules (1)

$$\frac{P_1 \text{ precise} \qquad \{P_1\}\ C\ \{P_2\} \qquad x \notin \mathsf{fv}(P_1 * P_3) \qquad \mathsf{Thread}(x, P_2) * P_3 \Rightarrow P_4 \quad \mathsf{allowed}([\![x := *]\!], P_3)}{\{P_1 * P_3\}\ x := \mathbf{fork}_{[P_1,P_2]}\ C\ \{P_4\}} \ (\text{FORK})$$

consequence built in due to stability

$$\frac{}{\{P * \mathsf{Thread}(E, P')\}\ \mathbf{join}\ E\ \{P * P'\}} \ (\text{JOIN})$$

Thread(E,P’) may not be stable, so can’t use frame rule; but the stability of the postcondition follows from that of the precondition

The Rules (2)

- Implicit assumption: any assertion is stable.

  $$if\ \sigma, pr, \gamma \models P\ and\ (\sigma, \sigma') \in pr.R,\ then\ \sigma', pr, \gamma \models P$$

  (so *pr*'s are trivially stable)

- Writes must be allowed.

  $$allowed(K, P) \iff$$

  $$if\ \sigma, pr, \gamma \models P\ and\ (\sigma, \sigma') \in K,\ then\ (\sigma, \sigma') \in pr.G.$$

The Rules (3)

$$\frac{P_1 \Rightarrow P_1' \quad \{P_1'\}\, C\, \{P_2'\} \quad P_2' \Rightarrow P_2}{\{P_1\}\, C\, \{P_2\}}\ (\textsc{cons})$$

$$\frac{\{P\}\, C\, \{P'\} \quad \mathsf{stable}(P_0)}{\{P * P_0\}\, C\, \{P' * P_0\}}\ (\textsc{frame})$$

$$\frac{P \Rightarrow [E/x]P' \quad \mathsf{allowed}([\![x := \underline{E}]\!], P)}{\{P\}\ x := E\ \{P'\}}\ (\textsc{assn})$$

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Soundness: Definitions

$\models \{P\}C\{Q\}$ *asserts that, if* $\sigma, pr, \gamma \models P$, *then*

(1) $\vdash (C, \sigma, pr, \gamma)$ safe; *and*

(2) if $(C, \sigma, pr, \gamma) \leadsto^* (\mathbf{skip}, \sigma', pr', \gamma')$,

then $\sigma', pr', \gamma' \models Q$.

Includes interference steps

$$
\begin{aligned}
\vdash \mathbf{fork}_{[P,Q]}\, C \ \mathsf{wa} &\iff \models \{P\}C\{Q\} \ \wedge \ \vdash C \ \mathsf{wa} \\
\vdash \mathbf{skip}\ \mathsf{wa} &\iff \mathit{always} \\
\vdash C_1; C_2 \ \mathsf{wa} &\iff \vdash C_1 \ \mathsf{wa} \ \wedge \ \vdash C_2 \ \mathsf{wa} \\
&\ldots
\end{aligned}
$$

Soundness Theorems

- **Local soundness:**

If $\vdash \{P\}C\{Q\}$, *then* $\models \{P\}C\{Q\}$ *and* $\vdash C$ wa.

- **Global soundness:**

If $\vdash \{P\}C\{Q\}$ *and* $\sigma, 1, \emptyset \models P$, *then*

– $\neg((\sigma, [t \mapsto C, 1, \emptyset]) \Longrightarrow^* \textbf{abort})$; *and*

– *if* $(\sigma, [t \mapsto C, 1, \emptyset]) \Longrightarrow^* (\sigma', [t \mapsto \textbf{skip}, pr, \gamma])$ *then* $\sigma', pr, \gamma \models Q$.

Includes interference steps

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Example (ver 1)

```
{ full}
        t1 := fork (x := 1;)

        t2 := fork (x := 2;)

        join t1;

        x := 2;

        join t2;
{ x = 2 }
```

Example (ver 1)

T1 = $[\mathbf{x} : Z \to \{1\}]_1$
G2 = $[\mathbf{x} : Z \to \{2\}]_1$

{T1} **x := 1** {T1}
{G2} **x := 2** {G2}

{ full } { (full – T1) * T1 }

t1 := fork (x := 1;)

{ (full – T1) * Thread(t1, T1) }

t2 := fork (x := 2;)

{ (full – T1 – G2) * Thread(t1, T1) * Thread(t2, G2) }

join t1;

{ (full – G2) * Thread(t2, G2) }

x := 2;

{ ??? }

join t2;

{ x = 2 }

Example (ver 1)

T1 = $[\mathbf{x} : Z \to \{1\}]_1$
G2 = $[\mathbf{x} : Z \to \{2\}]_{0.5g}$

{G1} **x := 1** {G1}
{G2} **x := 2** {G2}

{ full } { (full – T1) * T1 }

t1 := fork (x := 1;)

{ (full – T1) * Thread(t1, T1) }

t2 := fork (x := 2;)

{ (full – T1 – G2 – G2) * Thread(t1, T1) * G2 * Thread(t2, G2) }

join t1;

{ (full – G2 – G2) * G2 * Thread(t2, G2) }

x := 2;

{ (full – G2 – G2) * G2 * Thread(t2, G2) * x = 2 }

join t2;

{ x = 2 }

Need to check stability!

Example (ver 2)

$\{T_1 * G_2 * G_2 * D_3 * D_3 * L' * \mathtt{x} \neq 1\}$

$\quad$ `t1 := fork`$_{[T_1*(\mathtt{x}\neq 1), T_1]}$ `(if(x==1) error; x := 1)`

$\{G_2 * G_2 * D_3 * D_3 * L' * \mathsf{Thread}(\mathtt{t1}, T_1)\}$

$\quad$ `t2 := fork`$_{[G_2*D_3, G_2*D_3]}$ `(x := 2; if(x==3) error)`

$\{G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t1}, T_1) * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3)\}$

$\quad$ `join t1;`

$\{T_1 * G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3)\}$

$\quad$ `x := 2;`

$\{T_1 * G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3) * \mathtt{x} = 2\}$

$\quad$ `join t2;`

$\{T_1 * G_2 * G_2 * D_3 * D_3 * L' * \mathtt{x} = 2\}$

where $T_1 \stackrel{\mathbf{def}}{=} [\mathtt{x} : \mathbb{Z} \rightsquigarrow 1]_1$, $G_2 \stackrel{\mathbf{def}}{=} [\mathtt{x} : \mathbb{Z} \rightsquigarrow 2]_{\frac{1}{2}\mathbf{g}}$, $D_3 \stackrel{\mathbf{def}}{=} [\mathtt{x} : \mathbb{Z} \rightsquigarrow 3]_{\frac{1}{2}\mathbf{d}}$,

and $L' \stackrel{\mathbf{def}}{=} [\mathtt{x} : \mathbb{Z} \rightsquigarrow \{1, 2, 3\}]_1 \mathrel{-\!\!*} \mathsf{full}$

Example (ver 2)

- Motivating example in the paper: R-G proof requires auxiliary state!
- But same is true for D-G when we make a simple change to previous example.

Example (ver 3)

$\{T_1 * G_2 * G_2 * D_3 * D_3 * L' * \mathtt{x} \neq 1\}$

$\quad \mathtt{t1\ :=\ fork}_{[T_1*(\mathtt{x}\neq 1),T_1]}\ \mathtt{(if(x==1)\ error;\ x\ :=\ 1)}$

$\{G_2 * G_2 * D_3 * D_3 * L' * \mathsf{Thread}(\mathtt{t1}, T_1)\}$ $\boxed{\mathbf{x := 3;\ x := 2;}}$

$\quad \mathtt{t2\ :=\ fork}_{[G_2*D_3,G_2*D_3]}\ \mathtt{(}\cancel{\mathtt{x\ :=\ 2;}}\ \mathtt{if(x==3)\ error)}$

$\{G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t1}, T_1) * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3)\}$

$\quad \mathtt{join\ t1;}$

$\{T_1 * G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3)\}$

$\quad \mathtt{x\ :=\ 2;}$

$\{T_1 * G_2 * D_3 * L' * \mathsf{Thread}(\mathtt{t2}, G_2 * D_3) * \mathtt{x} = 2\}$

$\quad \mathtt{join\ t2;}$

$\{T_1 * G_2 * G_2 * D_3 * D_3 * L' * \mathtt{x} = 2\}$

where $T_1 \stackrel{\mathbf{def}}{=} [\mathtt{x}\colon \mathbb{Z} \rightsquigarrow 1]_1$, $G_2 \stackrel{\mathbf{def}}{=} [\mathtt{x}\colon \mathbb{Z} \rightsquigarrow 2]_{\frac{1}{2}\mathbf{g}}$, $D_3 \stackrel{\mathbf{def}}{=} [\mathtt{x}\colon \mathbb{Z} \rightsquigarrow 3]_{\frac{1}{2}\mathbf{d}}$,

and $L' \stackrel{\mathbf{def}}{=} [\mathtt{x}\colon \mathbb{Z} \rightsquigarrow \{1,2,3\}]_1 \mathrel{-\!\!*} \mathsf{full}$

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Encoding R-G

See paper.

Outline

- Raw language semantics
- Permissions and their meaning
- Instrumented language semantics
- Logic rules
- Soundness
- Example
- Encoding of Rely-Guarantee
- Related work

Related Work

- Fork/join generally ignored
- Feng et al., Hobor et al.: no join, argue that threads can synchronize explicitly. Not compositional: interference must be specified globally
- Gotsman et al. (storable locks): “this is achieved by defining an invariant over protected sections of the heap, which makes compositional reasoning about inter-thread interference impossible”