RGsep and Local Rely-Guarantee: An example

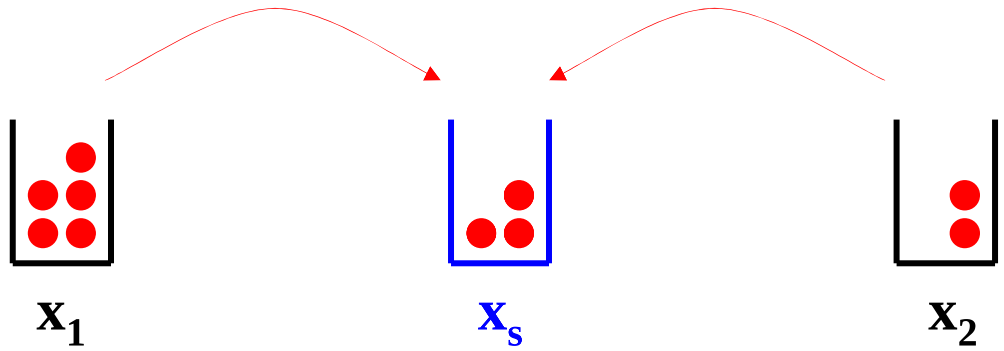
Arthur Charguéraud

Concurrent Program Logics Seminar

Max Planck Institute for Software Systems Kaiserslautern, 2011/07/04

Example: parallel increments

Scenario:



– the program starts with a pair of two natural numbers (n_1,n_2)

– the program allocates a shared memory cell and starts two threads

– the first thread has a private cell containing the value n_1

– the second created has a private cell containing the value n_2

– both threads do atomic unit transfer from their cell to the shared cell

– they stop when their private cell contains the value zero

– we want to prove that in the end the shared cell contains n_1+n_2

Initial situation

Initial state involves three variables: (only showing pre-conditions)

Emp; Emp; emp |- $(x1 \rightarrow n1) * (x2 \rightarrow n2) * (xs \rightarrow 0)$

rely, guarantee, and fence invariant

We then introduce auxiliary variables representing the number of transfers:

Emp; Emp; emp |- $(x1 \rightarrow n1\text{-}m1) * (x2 \rightarrow n2\text{-}m2) * (xs \rightarrow m1+m2)$
$* (y1 \rightarrow m1) * (y2 \rightarrow m2) * (m1 \leq n1) * (m2 \leq n2)$

m1 = number of transfers made by thread 1

m2 = number of transfers made by thread 2

Recall the rule for parallel+sharing

$$\frac{\begin{array}{c}(R \vee G_2) * G_2'; G_1 * G_1'; I * I' \vdash \{p_1 * m * r\}\ C_1\ \{q_1 * m_1' * r_1'\} \\ (R \vee G_1) * G_1'; G_2 * G_2'; I * I' \vdash \{p_2 * m * r\}\ C_2\ \{q_2 * m_2' * r_2'\} \\ I \rhd \{R, G_1, G_2\} \quad I' \rhd \{G_1', G_2'\} \quad r \vee r_1' \vee r_2' \Rightarrow I \quad m \vee m_1' \vee m_2' \Rightarrow I'\end{array}}{R; G_1 \vee G_2; I \vdash \{p_1 * p_2 * m * r\}\ C_1 \parallel C_2\ \{q_1 * q_2 * (m_1' \wedge m_2') * (r_1' \wedge r_2')\}}\ (\text{PAR-HIDE})$$

$\rightarrow$ for the parent, $\mathbf{p_1}$ and $\mathbf{p_2}$ and **m** are private, and **r** is shared

$\rightarrow$ for the left branch, $\mathbf{p_1}$ is private and **m** and **r** are shared

$\rightarrow$ for the right branch, $\mathbf{p_2}$ is private and **m** and **r** are shared

$\rightarrow$ **I** is the fence for **r** and **I'** is the fence for **m**

Application of the rule on the example

Apply the par-hide rule to produce two subgoals:

R2; G1; I' |- {(x1 → n1-m1) ∗ (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2)} P1
{... ∗ (m1 = n1) }

shared resource (**r**)

R1; G2; I' |- {(x2 → n2-m2) ∗ (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2)} P2
{... ∗ (m2 = n2) }

sufficient to conclude that
xs → n1+n2 in the end

Using the following instantiations:

I = **empty**

R = **Empty**

I' = **∃m1. ∃m2. (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2)**

G1 = R2 = ∃m1. (∃m2. (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2)
~> ∃m2. (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2))

G2 = R1 = symmetrically defined

And checking the side-condition: **I' ►{R1,R2,G1,G2}**

Bugfix: upper bound on the transfers

Apply the par-hide rule to produce two subgoals:

R2; G1; I' |- {(x1 $\to$ n1-m1) $*$ (xs $\to$ m1+m2) $*$ (y1 $\to$ m1) $*$ (y2 $\to$ m2) $*$ (m1 $\leq$ n1)} P1
{... $*$ (m1 = n1) }

shared resource (**r**)

R1; G2; I' |- {(x2 $\to$ n2-m2) $*$ (xs $\to$ m1+m2) $*$ (y1 $\to$ m1) $*$ (y2 $\to$ m2) $*$ (m2 $\leq$ n2)} P2
{... $*$ (m2 = n2) }

sufficient to conclude that
xs $\to$ n1+n2 in the end

Using the following instantiations:

I = empty

R = Empty

I' = $\exists$m1. $\exists$m2. (xs $\to$ m1+m2) $*$ (y1 $\to$ m1) $*$ (y2 $\to$ m2) $*$ (m1 $\leq$ n1) $*$ (m2 $\leq$ n2)

G1 = R2 = $\exists$m1. ($\exists$m2. (xs $\to$ m1+m2) $*$ (y1 $\to$ m1) $*$ (y2 $\to$ m2) $*$ (m2 $\leq$ n2)
~> $\exists$m2. (xs $\to$ m1+m2) $*$ (y1 $\to$ m1) $*$ (y2 $\to$ m2) $*$ (m2 $\leq$ n2))

G2 = R1 = symmetrically defined

And checking the side-condition: **I' ►{R1,R2,G1,G2}**

A weaker fence

Instead of a fence that captures a connection between **xs, y1** and **y2**:

$$I' = \exists m1.\ \exists m2.\ (xs \rightarrow m1+m2) * (y1 \rightarrow m1) * (y2 \rightarrow m2)$$

we can use a weaker invariant covering only the existence of the cells:

$$I' = (xs \rightarrow -) * (y1 \rightarrow -) * (y2 \rightarrow -)$$

which is short for:

$$I' = \exists ms.(xs \rightarrow ms) * \exists m1.(y1 \rightarrow m1) * \exists m2.(y2 \rightarrow m2)$$

$\rightarrow$ Intuitively, the fence only needs to cover the footprint of the shared state.

Example in RG-sep

Apply the par rule to produce two subgoals:

R2; G1 |- {(x1 → n1-m1) ∗ [(xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2) ∗ (m1 ≤ n1)]} P1
{... ∗ (m1 = n1) }

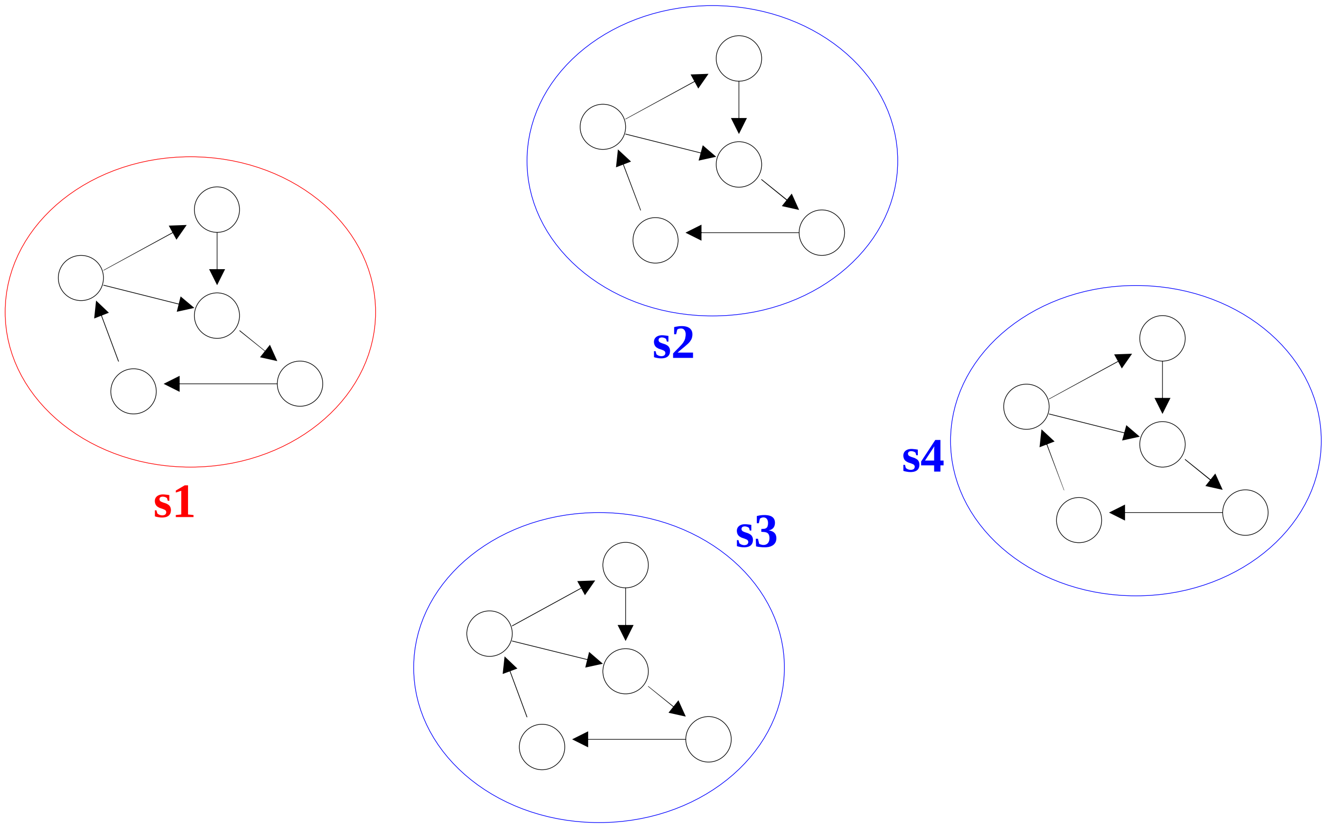
R1; G2 |- {(x2 → n2-m2) ∗ [(xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2) ∗ (m2 ≤ n2)]} P2
{... ∗ (m2 = n2) }

where:

G1 = R2 = (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2) ∗ (m2 ≤ n2)
~> (xs → m1+m2) ∗ (y1 → m1) ∗ (y2 → m2) ∗ (m2 ≤ n2)
G2 = R1 = symmetrically defined

Rely-guarantee viewed as an automaton

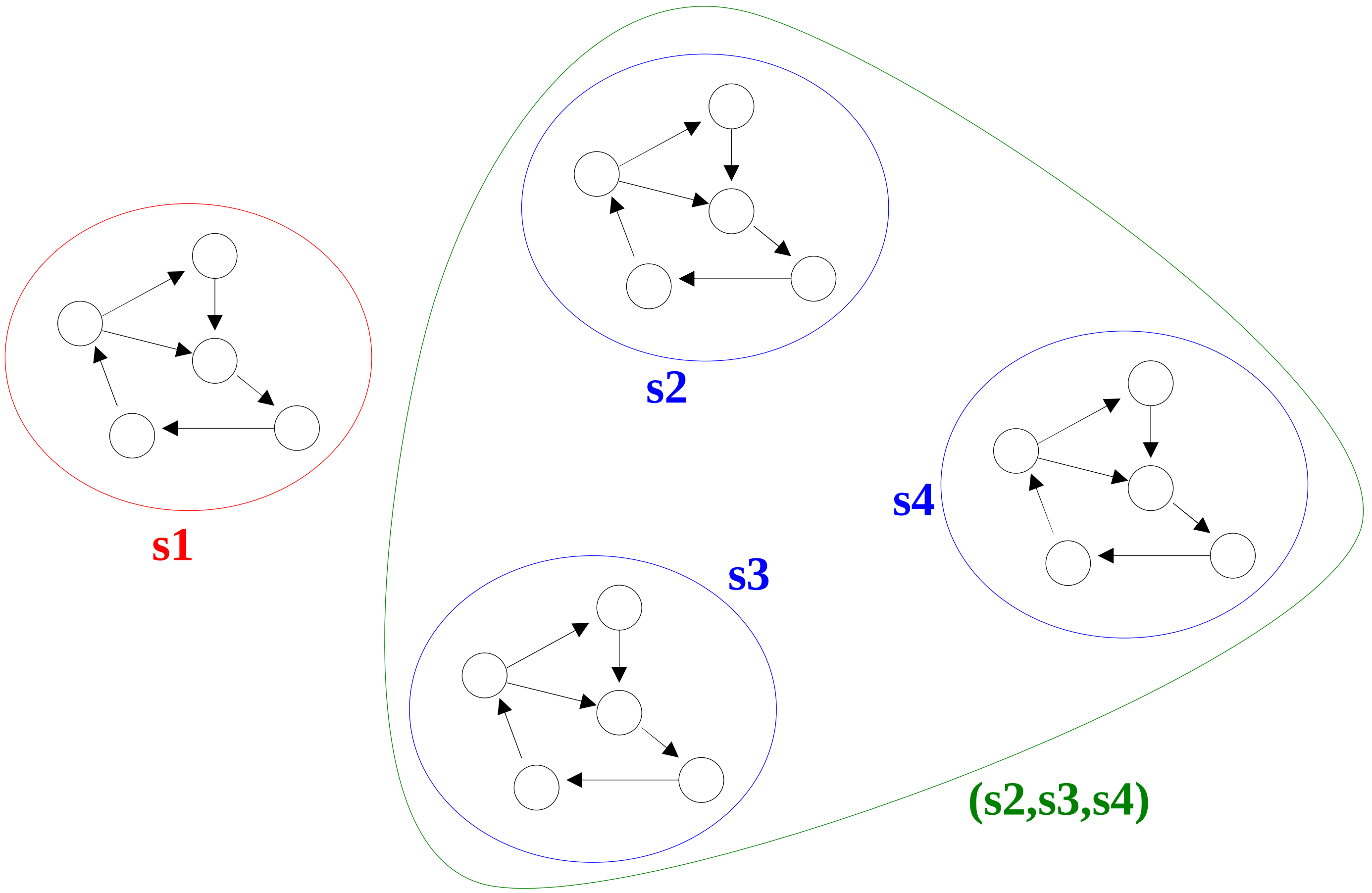
View a concurrent system as an automaton (a state machine with transitions). Each thread correspond to a state of one automaton.



Rely-guarantee viewed as an automaton

View a concurrent system as an automaton (a state machine with transitions). Each thread correspond to a state of one automaton.

The "context" of a thread is the product state of all the other threads.



Rely-guarantee viewed as an automaton

To verify one thread, we use:

1) an auxiliary variable to capture the state of this thread **(y1 $\rightarrow$ m1)**

2) an auxiliary variable to capture the state of other threads **(y2 $\rightarrow$ m2)**

3) a description of the private data **(x1 $\rightarrow$ n1-m1)**

4) a description of the shared data **(xs$\rightarrow$ m1+m2)**

Remarks:

$\rightarrow$ (2) corresponds to the product of the states of all the concurrent threads

$\rightarrow$ using (1) and (2) suffices to specify the state of the entire system

$\rightarrow$ the content of shared data (4) depends on the state of the entire system

$\rightarrow$ the content of the private data (3) may depend only on the local state (1) but in general it would also depend on the description of the global state (2)