CSim^2: Compositional Top-down Verification of Concurrent Systems using Rely-Guarantee

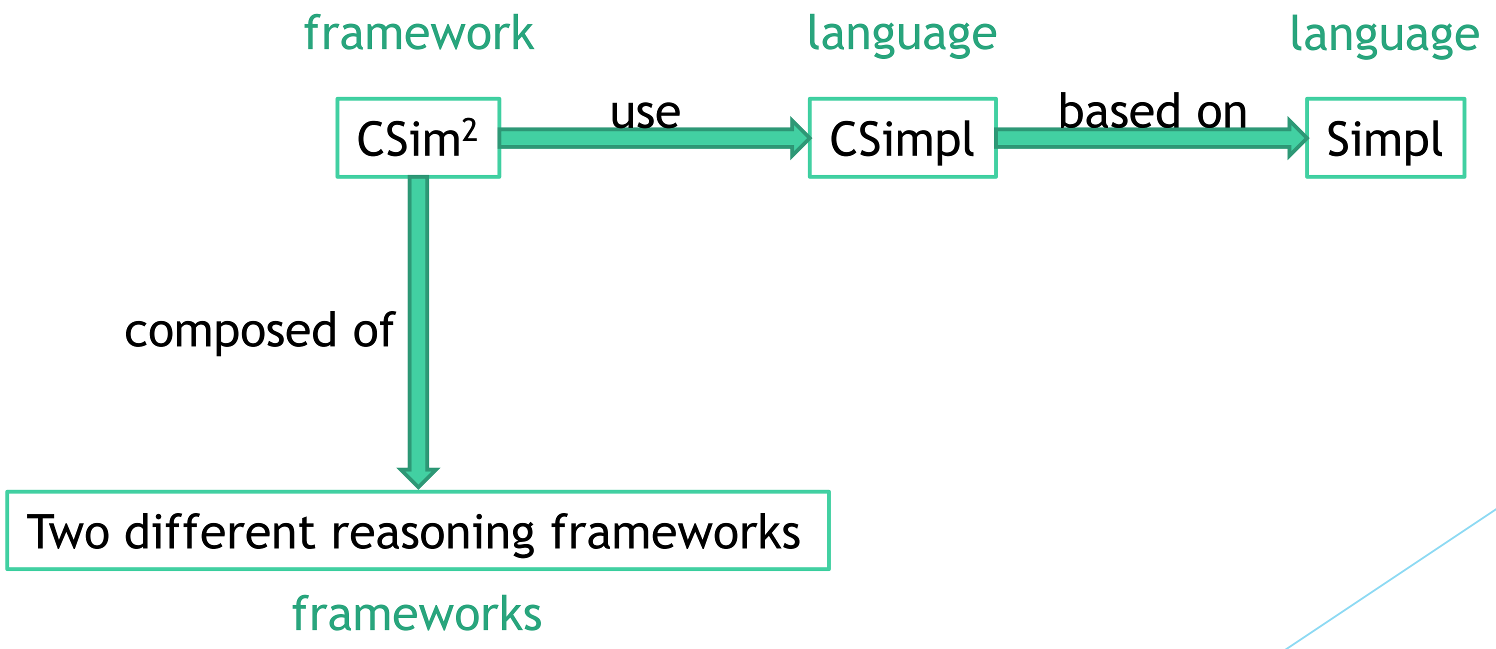
DAVID SANAN, YONGWANG ZHAO, SHANG-WEI LIN and LIU YANG

报告人：赵鸿燕

- 1. Introduction
- 2. CSIM^2 Overview
- 3. Syntax and Semantics of CSimpl Language
- 4. Rely-Guarantee for CSimpl
- 5. Simulation and Property Preservation
- 6. Case Study

1. Introduction
main concepts

- 1. Top-down verification uses step-wise verification between different layers of refinement, where verification of properties is conducted on the top layers representing high specification levels. Then, those verified properties are propagated down to the bottom layers representing low specification levels.
- 2.



2. CSIM^2 Overview
CSim^2 Architecture

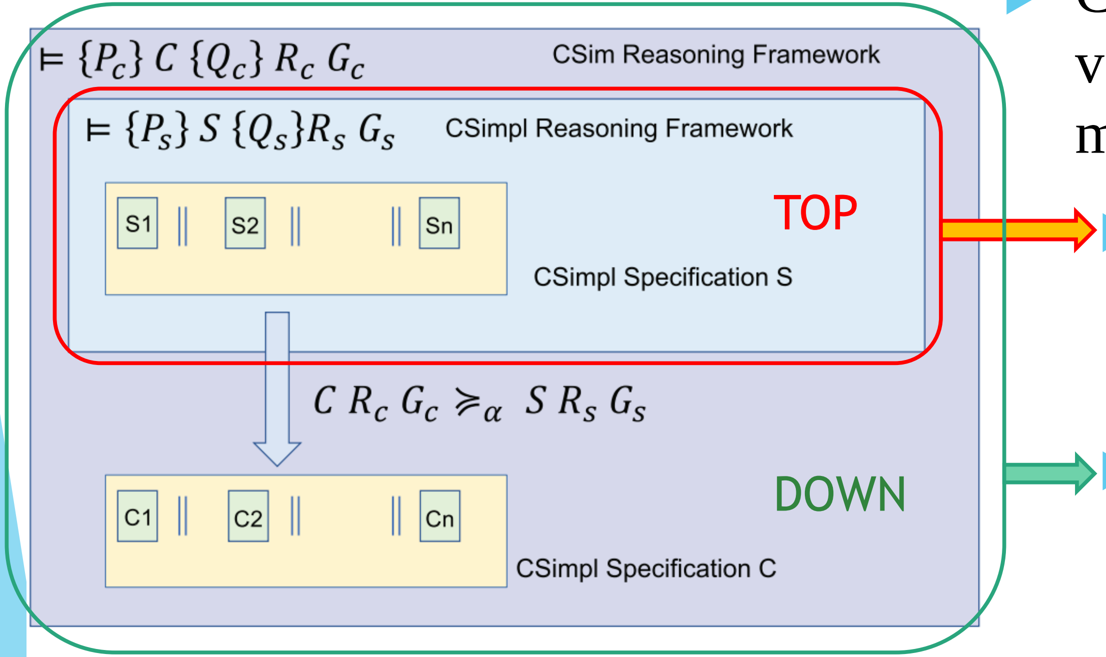


Fig. 1. **CSim^2** Architecture.

- CSim^2 allows a top-down verification approach to verify properties on concurrent specifications by means of two different reasoning frameworks:
  - a rely-guarantee reasoning framework for the verification of properties in <u>concurrent CSimpl specifications</u>;
  - and a simulation-based rely-guarantee property preservation framework, to show that properties proven in an abstraction layer <u>are preserved on</u> lower layers.

2. CSIM^2 Overview
A Rely-Guarantee Reasoning Framework

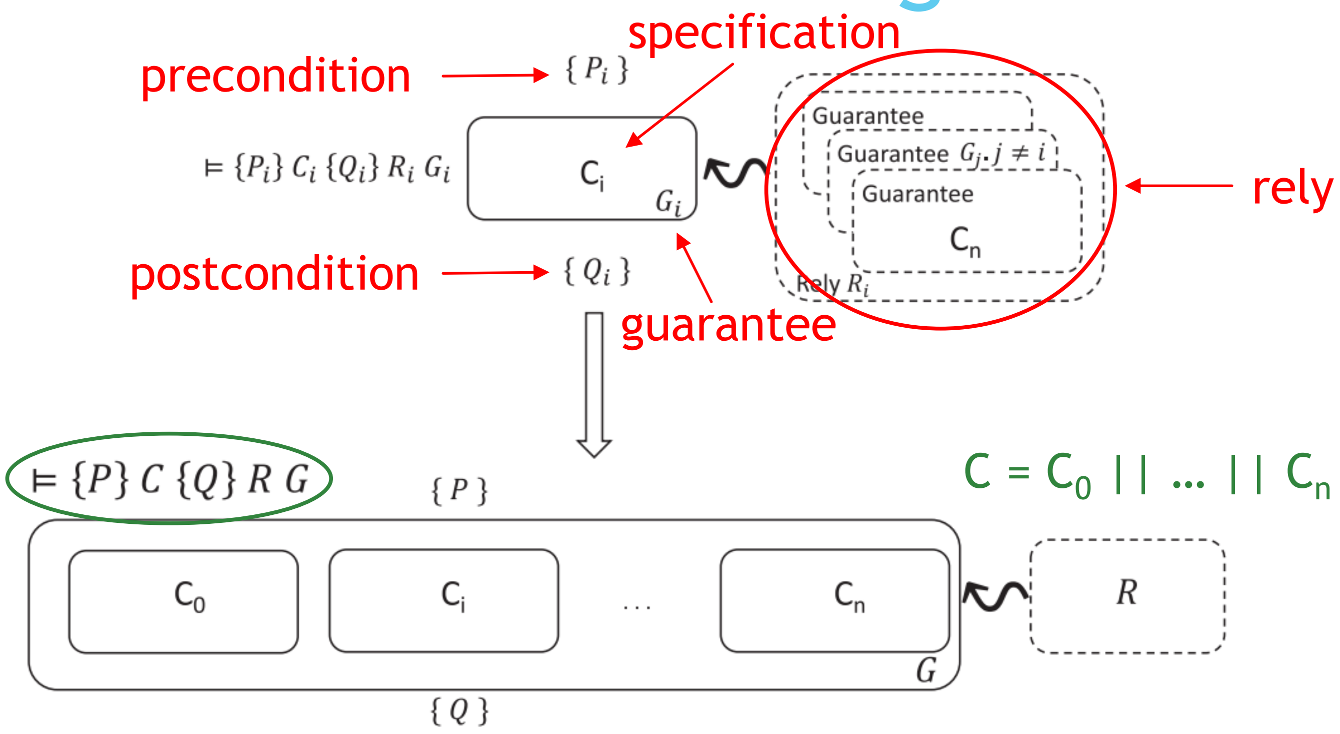


Fig. 2. **CSim^2** Property Verification Compositionality.

- CSim^2 provides sound reasoning rules for all the constructs of the specification language and a parallel compositional rule for the verification of parallel systems.

2. CSIM^2 Overview

A Simulation-Based Rely-Guarantee Property Preservation Framework

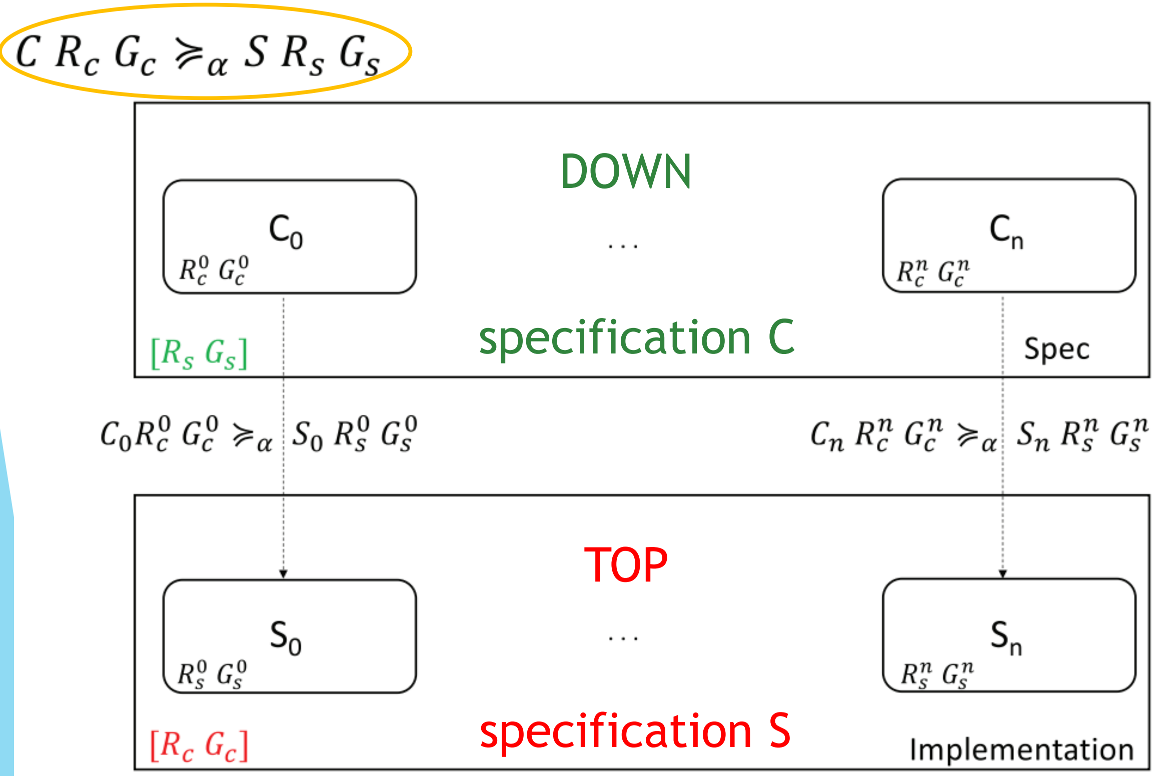


Fig. 3. CSim^2 Simulation Compositionality.

$|=_p \{ P_s \}\ S\ \{ Q_s \}\ R_s\ G_s$

- Prove $|=_p \{ P_c \}\ C\ \{ Q_c \}\ R_c\ G_c$ by showing that:
  - (1) given a simulation relation α, and the parallel specification S with rely and guarantee relations R_s and G_s then C, with the relations R_c and G_c , is an implement-tation of S in α, denoted as

    $$\boxed{C\ R_c\ G_c \succcurlyeq_\alpha S\ R_s\ G_s}$$

  - (2) $P_c \subseteq \alpha`P_s$ and $Q_c \subseteq \alpha`Q_s$, where ` is the image operation over a relation and a set.

3. Syntax and Semantics of CSimpl Language

CSimpl Syntax

状态 程序名 失败状态

type-synonym *'s bexp = 's set*
datatype *('s, 'p, 'f) com =*
Skip | *Throw* | *Basic 's* $\Rightarrow$ *'s* | *Spec ('s* $\times$ *'s) set* | *Seq ('s ,'p, 'f) com ('s,'p, 'f) com*
| *Cond 's bexp ('s,'p,'f) com ('s,'p,'f) com* | *While 's bexp ('s,'p,'f) com* | *Call 'p*
| *DynCom 's* $\Rightarrow$ *('s,'p,'f) com* | *Guard 'f 's bexp ('s,'p,'f) com*
| *Catch ('s,'p,'f) com ('s,'p,'f) com* | *Await 's bexp ('s,'p,'f) scom*
datatype *('s,'f) xstate = Normal 's* | *Abrupt 's* | *Fault 'f* | *Stuck*
type-synonym *('s,'p,'f) config = ('s,'p,'f) com* $\times$ *('s,'f) xstate*
type-synonym *('s,'p,'f) body = 'p* $\Rightarrow$ *('s,'p,'f) com option*
type-synonym *('s,'p,'f) par-Simpl = ('s,'p,'f) com list*

Fig. 4. Syntax and state definition of the CSimpl Language.

3. Syntax and Semantics of CSimpl Language
CSimpl Syntax

失败断言，'f是可能的错误范围

正常状态

异常状态

datatype $('s,'f)$ *xstate* = *Normal* 's | *Abrupt* 's | *Fault* 'f | *Stuck*

1.调用未定义函数
2.程序是Spec r，但状态不在不确定关系r内

For simplicity on the notation, we represent the `xstates` `Normal` σ, `Abrupt` σ, and `Fault F` with $\sigma_{\uparrow N}$, $\sigma_{\uparrow A}$, and $\sigma_{\uparrow F}$. We use $\uparrow^N \sigma$ to represent the predicate $\exists \sigma_n . \sigma = \texttt{Normal}\ \sigma_n$, $\uparrow^A \sigma$ to represent the predicate $\exists \sigma_a . \sigma = \texttt{Abrupt}\ \sigma_a$, and $\uparrow^F \sigma$ to represent the predicate $\exists \texttt{f} . \sigma = \texttt{Fault f}$. Throughout this article, we will refer to the type `('σ, 'f)xstate` as $'\sigma$ when is clear in the context.

3. Syntax and Semantics of CSimpl Language

CSimpl Syntax

正常终止

由关系r表示的不确定行为

$$Skip \mid Throw \mid Basic\ \boxed{'s \Rightarrow\ 's} \mid Spec\ \boxed{('s \times\ 's)\ set} \mid Seq\ \boxed{('s\ ,'p,\ 'f)\ com}\ \boxed{('s,'p,\ 'f)\ com}$$

异常终止

接收一个用于修改状态的函数

Seq S1 S2

3. Syntax and Semantics of CSimpl Language

CSimpl Syntax

Cond b C1 C2

While b C

唤醒程序p，否则状态改为Stuck

$| \ Cond \ \boxed{'s \ bexp} \ \boxed{('s,'p,'f) \ com} \ \boxed{('s,'p,'f) \ com} \ | \ While \ \boxed{'s \ bexp} \ \boxed{('s,'p,'f) \ com} \ | \ Call \ \boxed{'p}$

$| \ DynCom \ \boxed{'s \Rightarrow ('s,'p,'f) \ com} \ | \ Guard \ \boxed{'f} \ \boxed{'s \ bexp} \ \boxed{('s,'p,'f) \ com}$

依赖于状态的动态命令

Guard f g C，如果guard g成立则执行C，否则状态改为f

3. Syntax and Semantics of CSimpl Language

CSimpl Syntax

Catch C1 C2，执行C1，若C1 Throw，则执行C2

$$| \ Catch \ \boxed{('s,'p,'f) \ com} \ \boxed{('s,'p,'f) \ com} \ | \ Await \ \boxed{'s \ bexp} \ \boxed{('s,'p,'f) \ scom}$$

Await b body

- Await allows synchronization of processes on a predicate b over shared variables and it atomically executes body.

3. Syntax and Semantics of CSimpl Language

Simpl Syntax

CSimpl extends Simpl by adding two constructs for concurrency: Await and Parallel Composition.

type-synonym *'s bexp = 's set*
datatype *('s, 'p, 'f) com =*
Skip | Throw | Basic 's ⇒ 's | Spec ('s × 's) set | Seq ('s ,'p, 'f) com ('s,'p, 'f) com
| Cond 's bexp ('s,'p,'f) com ('s,'p,'f) com | While 's bexp ('s,'p,'f) com | Call 'p
| DynCom 's ⇒ ('s,'p,'f) com | Guard 'f 's bexp ('s,'p,'f) com
| Catch ('s,'p,'f) com ('s,'p,'f) com | Await 's bexp ('s,'p,'f) scom
datatype *('s,'f) xstate = Normal 's | Abrupt 's | Fault 'f | Stuck*
type-synonym *('s,'p,'f) config = ('s,'p,'f) com × ('s,'f) xstate*
type-synonym *('s,'p,'f) body = 'p ⇒ ('s,'p,'f) com option*
type-synonym *('s,'p,'f) par-Simpl = ('s,'p,'f) com list*

Fig. 4. Syntax and state definition of the CSimpl Language.

3. Syntax and Semantics of CSimpl Language
Simpl Syntax

- Some examples of the concrete syntax of Simpl are given by:

- `C1;;`$_s$`C2` represents `Seq C1 C2`.
- `IF`$_s$ `b THEN C1 ELSE C2 FI` represents `Cond b C1 C2`.
- `WHILE`$_s$ `b DO C OD` represents `While b C`. A variance `WHILE`$_s$`b DO INV I C OD` represents the loop annotated with the invariant `I`.
- $\boxed{\texttt{´f:==}_s\texttt{v} \text{ represents } \texttt{Basic}(\lambda s.\ s(\!|f\_':=\ v|\!))}$. `´f` is used to represent the selects and updates of fields of the state when the state is represented by a record.

3. Syntax and Semantics of CSimpl Language
big step and small step semantics

- The behavior of Simpl programs is defined in terms of big step and small step semantics.
- In the big step semantics,

$$\Gamma \vdash \langle C, \sigma \rangle \Rightarrow \sigma'$$

represents that in an environment for procedures Γ, the program C starting from the state σ reaches a final state σ'.

- The small step semantics uses a fine grain transition to carry out a single step.

$$\Gamma \vdash \langle C, \sigma \rangle \rightarrow \langle C', \sigma' \rangle$$

transitions in a step from the configuration <C, σ> to <C', σ'>.

3. Syntax and Semantics of CSimpl Language

partial and total correctness

初始状态是Normal状态

To reason about programs, Simpl includes a sound and partially complete reasoning framework for partial and total correctness of specifications based on a Floyd/Hoare-like logic. A Simpl specification is given by a program C in an environment for procedures Γ, and a precondition P and postconditions Q, A (for normal and abrupt postconditions). A specification is partially correct, which is denoted by $\Gamma \models \{P\}\, C\, \{Q\}, \{A\}$, when if $\Gamma \vdash \langle C, \sigma_{n\uparrow N} \rangle \Rightarrow \sigma'$ and $\sigma_n \in P$, then $\sigma' = \sigma'_{n\uparrow N}$ (σ' is a normal state) and $\sigma'_n \in Q$, or $\sigma' = \sigma'_{a\uparrow A}$ (σ' is an abrupt state) and $\sigma'_a \in A$. $\Gamma \models_t \{P\}\, C\, \{Q\}, \{A\}$ represents total correctness and it is satisfiable when in addition to partial correctness, C finishes for all possible executions starting from the precondition P. Note that although the postcondition often are expressed as relational postconditions relating the initial and final states, postconditions in Simpl and CSimpl only operate over final states.

终止状态是Abrupt状态

3. Syntax and Semantics of CSimpl Language
CSimpl Semantics

[Basic]

$$\frac{-}{\Gamma \vdash_c (Basic\ f, \boxed{\sigma_{\uparrow N}}) \rightarrow (Skip, \boxed{f\,\sigma_{\uparrow N}})}$$

[DynCom]

$$\frac{-}{\Gamma \vdash_c (DynCom\ c, \sigma_{\uparrow N}) \rightarrow (c\ \sigma, \sigma_{\uparrow N})}$$

[SpecC]

$$\frac{(\sigma, \sigma') \in r}{\Gamma \vdash_c (Spec\ r, \sigma_{\uparrow N}) \rightarrow (Skip, \sigma'_{\uparrow N})}$$

[SpecStuck]

$$\frac{\forall \sigma'.(\sigma, \sigma') \notin r}{\Gamma \vdash_c (Spec\ r, \sigma_{\uparrow N}) \rightarrow (Skip, \boxed{Stuck})}$$

[Guard]

$$\frac{\sigma \in g}{\Gamma \vdash_c (Guard\ f\ g\ p, \sigma_{\uparrow N}) \rightarrow (p, \sigma_{\uparrow N})}$$

[GuardFault]

$$\frac{\sigma \notin g}{\Gamma \vdash_c (Guard\ f\ g\ p, \sigma_{\uparrow N}) \rightarrow (Skip, \boxed{f_{\uparrow F}})}$$

3. Syntax and Semantics of CSimpl Language

CSimpl Semantics

[Seq]
$$\frac{\Gamma \vdash_c (c_1, \sigma) \rightarrow (c_1', \sigma')}{\Gamma \vdash_c (Seq\ c_1\ c_2, \sigma) \rightarrow (Seq\ c_1' c_2, \sigma')}$$

[SeqSkip]
$$\frac{-}{\Gamma \vdash_c (Seq\ Skip\ c_2, \sigma) \rightarrow (c_2, \sigma)}$$

[SeqThrow]
$$\frac{-}{\Gamma \vdash_c (Seq\ Throw\ c_2, \sigma_{\uparrow N}) \rightarrow (Throw, \sigma_{\uparrow N})}$$

[Catch]
$$\frac{\Gamma \vdash_c (c_1, \sigma) \rightarrow (c_1', \sigma')}{\Gamma \vdash_c (Catch\ c_1\ c_2, \sigma) \rightarrow (Catch\ c_1' c_2, \sigma')}$$

[CatchThrow]
$$\frac{-}{\Gamma \vdash_c \boxed{(Catch\ Throw\ c_2, \sigma_{\uparrow N}) \rightarrow (c_2, \sigma_{\uparrow N})}}$$

[CatchSkip]
$$\frac{-}{\Gamma \vdash_c (Catch\ Skip\ c_2, \sigma) \rightarrow (Skip, \sigma)}$$

3. Syntax and Semantics of CSimpl Language
CSimpl Semantics

[ConditionalTrue]
$$\frac{\sigma \in b}{\Gamma \vdash_c (Cond\ b\ c_1\ c_2, \sigma_{\uparrow N}) \rightarrow (c_1, \sigma_{\uparrow N})}$$

[ConditionalFalse]
$$\frac{\sigma \notin b}{\Gamma \vdash_c (Cond\ b\ c_1\ c_2, \sigma_{\uparrow N}) \rightarrow (c_2, \sigma_{\uparrow N})}$$

[While]
$$\frac{\sigma \in b}{\Gamma \vdash_c (While\ b\ c, \sigma_{\uparrow N}) \rightarrow (Seq\ c\ (While\ b\ c), \sigma_{\uparrow N})}$$

[WhileEnd]
$$\frac{\sigma \notin b}{\Gamma \vdash_c (While\ b\ c, \sigma_{\uparrow N}) \rightarrow (Skip, \sigma_{\uparrow N})}$$

Some c表示环境中定义了该程序　　禁止套娃

[Call]
$$\frac{\boxed{\Gamma\ p = Some\ c} \quad \boxed{c \neq Call\ p}}{\Gamma \vdash_c (Call\ p, \sigma_{\uparrow N}) \rightarrow (c, \sigma_{\uparrow N})}$$

p是被唤醒程序的名字

None表示环境中没有定义该程序

[CallUndefined]
$$\frac{\boxed{\Gamma\ p = None}}{\Gamma \vdash_c (Call\ p, \sigma_{\uparrow N}) \rightarrow (Skip, Stuck)}$$

3. Syntax and Semantics of CSimpl Language

CSimpl Semantics

由于大语义是顺序执行的，所以在环境中去掉使用Await的程序

σ’是Abrupt外的σ’

σ’是Abrupt

[AWAIT]

$$\frac{s \in b \quad \boxed{\Gamma_{\neg a}} \vdash \langle c, \sigma_{\uparrow N} \rangle \Rightarrow \sigma' \quad \boxed{\neg(\uparrow^{A} \sigma')}}{\Gamma \vdash_c (Await\ b\ p, \sigma_{\uparrow N}) \rightarrow (Skip, \sigma')}$$

[AWAITABRUPT]

$$\frac{s \in b \quad \Gamma_{\neg a} \vdash \langle c, \sigma_{\uparrow N} \rangle \Rightarrow \boxed{\sigma'_{\uparrow A}}}{\Gamma \vdash_c (Await\ b\ p, \sigma_{\uparrow N}) \rightarrow \boxed{(Throw, \sigma'_{\uparrow N})}}$$

[PAR]

$$\frac{i \leq n \quad \Gamma \vdash_c (C_i, \sigma) \rightarrow (C'_i, \sigma')}{\Gamma \vdash_p ([C_1 || \ldots \boxed{C_i} \ldots || C_n], \sigma) \rightarrow ([C_1 || \ldots \boxed{C'_i} \ldots || C_n], \sigma')}$$

不用Abrupt，而用
<Throw, Normal state>
表示异常终止

3. Syntax and Semantics of CSimpl Language

CSimpl Semantics

[Env]

$$\frac{-}{\Gamma \vdash_c (P, \sigma_{\uparrow N}) \boxed{\rightarrow_e} (P, \sigma')}$$

[Env_N]

$$\frac{\neg(\uparrow^N \sigma)}{\Gamma \vdash_c (P, \sigma) \rightarrow_e (P, \sigma)}$$

[P_Env]

$$\frac{-}{\Gamma \vdash_p ([C_1 || \ldots || C_n], \sigma_{\uparrow N}) \rightarrow_e ([C_1 || \ldots || C_n], \sigma'_{\uparrow N})}$$

对于并行程序，只能从一个Normal状态迁移到另一个Normal状态，
环境不能产生Fault状态给并行程序

3. Syntax and Semantics of CSimpl Language

CSimpl Semantics

- In the current version of CSimpl, the parallel construct is carried out in the top level similar to a multi-core architecture in which a number of programs is static, rather than providing a threadlike concurrency where it is possible to create new threads. There is not any technical issue in the CSimpl architecture to support a nested concurrency other than a more complex proof for soundness.

4. Rely-Guarantee for CSimpl

Rely and Guarantee

- Rely: how the environment interferes with the program
- Guarantee: how the program modifies the environment
- Therefore a property specification for the verification of parallel systems by using rely-guarantee is composed of five elements: the parallel system itself, a precondition, a postcondition, and a rely and guarantee relations.

4. Rely-Guarantee for CSimpl
Computation

Definition 4.1 (Sequential Component Computation). A computation is a tuple $(\Gamma, confs)$ where Γ is an environment for procedures and $confs = [(C_0, \sigma_1), (C_2, \sigma_2), \ldots, (C_n, \sigma_n)]$ is a list of sequential configurations. The set of possible computations $cptn$ is inductively defined as follows:

归纳定义：

- $(\Gamma, [(C, \sigma)]) \in cptn$
- if $\Gamma \vdash_c (C, \sigma) \rightarrow_e (C, \sigma')$ and $(\Gamma, (C, \sigma')\#xs) \in cptn$ then $(\Gamma, (C, \sigma)\#(C, \sigma')\#xs) \in cptn$
- if $\Gamma \vdash_c (C, \sigma) \rightarrow (C', \sigma')$ and $(\Gamma, (C', \sigma')\#xs) \in cptn$ then $(\Gamma, (C, \sigma)\#(C', \sigma')\#xs) \in cptn$

表示list l的第0个元素是(C, σ)

Definition 4.2 (Computations of an Initial Configuration). The set of possible computations of an initial configuration (C, σ) with an environment for procedures Γ, denoted as $cp\ (C, \sigma)\ \Gamma$, is the set of tuples (Γ, l) such that $\boxed{l!0 = (C, \sigma)}$ and $(\Gamma, l) \in cptn$.

The set of parallel computations par_{cp} is defined similar to cp by using parallel configurations and the semantic rules for parallel and environment step transitions defined by rules PAR and P_ENV in Figures 5 and 6.

4. Rely-Guarantee for CSimpl

Validity of Formulas for Rely-Guarantee in CSimpl

- By using the notion of computation, we define validity of a rely-guarantee tuple from the set of all possible computations from an initial configuration.
- It also uses the notions of assumption of the precondition and the environment, and commitment of the component and the postcondition.

4. Rely-Guarantee for CSimpl

Validity of Formulas for Rely-Guarantee in CSimpl

precondition p
rely R
environment Γ

Definition 4.3 (Validity Assumption). The assumption of a predicate p and an environment relation R, for an environment of procedures Γ, represented by $assum\ \Gamma\ p\ R$, is the set of component computations $(\Gamma, cptns)$ such that for any $[(C_0, \sigma_0), \ldots, (C_n, \sigma_n)] \in cptns$, with $n \geq 0$, then: (1) there exists a σ where $\sigma_0 = \sigma_{\uparrow N}$ and $\sigma \in p$, and (2) given a $k < n$ if there is an environment step transition $\Gamma \vdash_c (C_k, \sigma_k) \rightarrow_e (C_{k+1}, \sigma_{k+1})$, then $(\sigma_k, \sigma_{k+1}) \in R$.

4. Rely-Guarantee for CSimpl

Validity of Formulas for Rely-Guarantee in CSimpl

postcondition (q, a)
guarantee G
environment Γ
fault states F

Definition 4.4 (Validity Commitment). The commitment of a relation G, a pair of predicates (q, a), and a set of `Fault` states F, for an environment of procedures Γ, denoted as $comm\ \Gamma\ G\ (q, a)\ F$, is the set of component computations $(\Gamma, cptns)$ such that for any $[(C_0, \sigma_0), \ldots (C_n, \sigma_n)] \in cptns$, where $n \geq 0$ and there is not any f such that $\sigma_n = \mathtt{Fault}\ f$ and $f \in F$, then: (1) for any $k < n$, if $\Gamma \vdash_c (C_k, \sigma_k) \to (C_{k+1}, \sigma_{k+1})$, then $(\sigma_k, \sigma_{k+1}) \in G$; (2) if (C_n, σ_n) is a final configuration, then there is a σ' such that $\sigma_n = \sigma'_{\uparrow N}$ (2.1) and if $C_n = \mathtt{Skip}$ then $\sigma' \in q$ and if $C_n = \mathtt{Throw}$ then $\sigma' \in a$ (2.2).

4. Rely-Guarantee for CSimpl

Validity of Formulas for Rely-Guarantee in CSimpl

Definition 4.5 (Rely-Guarantee Validity). A specification of a component C w.r.t. a precondition p, a postcondition (q, a), a rely and guarantee relations R, and G, an environment of procedures Γ, and a set F of Faults, denoted as $\Gamma \models_{/F} P\ sat[p, R, G, q, a]$, is valid iff for all σ, $cp\ \Gamma\ C\ \sigma \cap assum(p, R) \subseteq comm(G, (q, a))\ F$.

Definition 4.6 (Rely-Guarantee CValidity). CValidity of a specification of a component C w.r.t. a precondition p, postcondition (q, a), a rely relation R, a guarantee relation G, an environment of procedures Γ, a specification of procedures Θ, and a set F of Faults, represented by $\Gamma, \Theta \models_{/F} P\ sat[p, R, G, q, a]$ iff if for any tuple $(c, p', R', G', q', a') \in \Theta\ \Gamma \models_{/F} (Call\ c)\ sat[p', R', G', q', a']$ is valid, then $\Gamma \models_{/F} P\ sat[p, R, G, q, a]$.

根据[28]，Θ是一组在验证过程中被认为是理所当然的程序规范的假设集合，它被用来处理递归程序

4. Rely-Guarantee for CSimpl

Validity of Formulas for Rely-Guarantee in CSimpl

THEOREM 4.7 (VALIDITY_COMPOSITIONALITY). *Given an environment for procedures* Γ, *a set of specifications for recursive procedures* Θ, *a rely-guarantee parallel specification is valid,*

$$\Gamma, \Theta \models_{/F} [C_0 || \ldots || C_n] SAT\ [p, R, G, q, a],$$

if for each $i \leq n$ *there exist* p_i, q_i, a_i, R_i, G_i, *representing a rely-guarantee specification for the i component such that*

(1) $\Gamma, \Theta \models_{/F} C_i sat[p_i, R_i, G_i, Q_i, A_i]$;
(2) $R \cup (\bigcup k \in \{k.\ k \leq n \wedge k \neq i\}.\ G_k \subseteq R_i)$;
(3) $\bigcup j \leq n.\ G_j \subseteq G$;
(4) $p \subseteq (\bigcap j \leq n.\ p_j)$;
(5) $(\bigcap j \leq n.\ q_j) \subseteq q$;
(6) $(\bigcup j \leq n.\ a_j) \subseteq a$.

4. Rely-Guarantee for CSimpl

Inference Rules of the Proof System

简单来说就是前/后置条件P去环境里逛了一圈回来还满足

- The intuition is that a precondition/postcondition P is valid before/after the execution of a sequential component C in a concurrent environment if the environment preserves P before/after the execution of C. Formally, with the relation R representing the concurrent environment:

P is stable when R holds.

Definition 4.8 (Stability). A set of states P is stable w.r.t. a relation R, represented by $Sta\ P\ R$, if given two states σ, σ', such that $\sigma \in P$ and $(\sigma, \sigma') \in R$, then $\sigma' \in P$.

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

换一种写法：p ⊆ { σ | f (σ) ∈ q }

[Basic]

$$\frac{\begin{array}{l} Sta\ p\ R \quad Sta\ q\ R \quad \boxed{p \subseteq \{\sigma.\ q\ (f\ \sigma)\}} \\ \forall \sigma\ \sigma'.\ \sigma \in p \wedge (\sigma' = f\ \sigma) \longrightarrow (\sigma_{\uparrow N}, \sigma'_{\uparrow N}) \in G \end{array}}{\Gamma, \Theta \vdash_{/F} Basic\ f\ \mathbf{sat}\ [p, R, G, q, a]}$$

[Throw]

$$\frac{Sta\ a\ R \quad \forall \sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} Throw\ \mathbf{sat}\ [a, R, G, q, a]}$$

[Spec] 由关系r表示的不确定行为

$$\frac{\begin{array}{l} Sta\ p\ R \quad Sta\ q\ R \\ p \subseteq \{\sigma.\ (\forall \sigma'.\ (\sigma, \sigma') \in r \longrightarrow q\ \sigma') \wedge (\exists \sigma'.\ (\sigma, \sigma') \in r)\} \\ \forall \sigma\ \sigma'. \sigma \in p \wedge (\sigma, \sigma') \in r \longrightarrow (\sigma_{\uparrow N}, \sigma'_{\uparrow N}) \in G \end{array}}{\Gamma, \Theta \vdash_{/F} Spec\ r\ \mathbf{sat}\ [p, R, G, q, a]}$$

[Skip]

$$\frac{Sta\ q\ R \quad \forall \sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} Skip\ \mathbf{sat}\ [q, R, G, q, a]}$$

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

[COND]

$$\dfrac{\begin{array}{l} Sta\ p\ R \quad \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G \\ \Gamma, \Theta \vdash_{/F}\ c1\ \textbf{sat}\ [p \cap b, R, G, q, a] \\ \Gamma, \Theta \vdash_{/F}\ c2\ \textbf{sat}\ [p \cap -b, R, G, q, a] \end{array}}{\Gamma, \Theta \vdash_{/F}\ Cond\ b\ c_1 c_2\ \textbf{sat}\ [p, R, G, q, a]}$$

[WHILE]

While程序的后置条件是p且非b

$$\dfrac{\begin{array}{l} Sta\ p\ R \quad Sta\ (\boxed{p \cap -b})\ R \quad Sta\ a\ R \\ \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G \\ \Gamma, \Theta \vdash_{/F}\ c\ \textbf{sat}\ [p \cap b, R, G, p, a] \end{array}}{\Gamma, \Theta \vdash_{/F}\ While\ b\ c\ \textbf{sat}\ [p, R, G, p \cap -b, a]}$$

[AWAIT]

$$\dfrac{\begin{array}{l} Sta\ p\ R \quad Sta\ q\ R \quad Sta\ a\ R \\ \forall\sigma.\ \Gamma_{\neg a}, \{\} \vdash_{/F} (p \cap b \cap \{\sigma\})c \\ \quad \{\sigma'.\ (\sigma_{\uparrow N}, \sigma'_{\uparrow N}) \in G\} \cap \{\sigma.q\ \sigma\}, \\ \quad \{\sigma'.\ (\sigma_{\uparrow N}, \sigma'_{\uparrow N}) \in G\} \cap \{\sigma.a\ \sigma\} \end{array}}{\Gamma, \Theta \vdash_{/F}\ Await\ b\ c\ \textbf{sat}\ [p, R, G, q, a]}$$

?

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

[CATCH]

$$\frac{\Gamma, \Theta \vdash_{/F} c1 \text{ sat } [p, R, G, q, \boxed{r}] \quad \Gamma, \Theta \vdash_{/F} c2 \text{ sat } [\boxed{r}, R, G, q, a] \quad Sta\ p\ R \quad Sta\ a\ R \quad \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} Catch\ c1\ c2 \text{ sat } [p, R, G, q, a]}$$

[SEQ]

$$\frac{\Gamma, \Theta \vdash_{/F} c1 \text{ sat } [p, R, G, \boxed{r}, a] \quad \Gamma, \Theta \vdash_{/F} c2 \text{ sat } [\boxed{r}, R, G, q, a] \quad Sta\ p\ R \quad Sta\ a\ R \quad \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} Seq\ c1\ c2 \text{ sat } [p, R, G, q, a]}$$

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

[GUARD]

$$\frac{\Gamma, \Theta \vdash_{/F} \ c \ \mathsf{sat} \ [p \cap g, R, G, q, a] \quad \boxed{Sta\ (p \cap g) R} \quad \forall \sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} \ Guard\ f\ g\ c \ \mathsf{sat} \ [p \cap g, R, G, q, a]}$$

[GUARD FAULT]

$$\frac{\Gamma, \Theta \vdash_{/F} \ c \ \mathsf{sat} \ [p, R, G, q, a] \quad \boxed{Sta\ p\ R \quad f \in F} \quad \forall \sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} \ Guard\ f\ g\ c \ \mathsf{sat} \ [p, R, G, q, a]}$$

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

[CALL]

$$\frac{\Gamma, \Theta \vdash_{/F} \ the(\Gamma\ c) \ \mathbf{sat}\ [p, R, G, q, a] \qquad Sta\ p\ R \quad c \in dom\ \Gamma \quad \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F} \ Call\ c\ \mathbf{sat}\ [p, R, G, q, a]}$$

[CALLREC]

$$\frac{\begin{array}{l}(c, p, R, G, q, a) \in Specs \\ \forall (c, p, R, G, q, a) \in Specs. \\ \quad c \in dom\ \Gamma \wedge Sta\ p\ R \wedge \forall\sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G \\ \quad \Gamma, \Theta \cup Specs \vdash_{/F} \ The(\Gamma\ c)\ \mathbf{sat}\ [p, R, G, q, a] \\ Sta\ p\ R \quad \forall\sigma.(\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G\end{array}}{\Gamma, \Theta \vdash_{/F} \ Call\ c\ \mathbf{sat}\ [p, R, G, q, a]}$$

[ASM]

$$\frac{(c, p, R, G, q, a) \in \Theta}{\Gamma, \Theta \vdash_{/F} \ Call\ c\ \mathbf{sat}\ [p, R, G, q, a]}$$

4. Rely-Guarantee for CSimpl

Rely-Guarantee proof rules for CSimpl

[DynCom]

$$\frac{\forall \sigma \in p.\ \Gamma, \Theta \vdash_{/F}\ c\ \sigma\ \textbf{sat}\ [p, R, G, q, a] \qquad Sta\ p\ R \quad \forall \sigma.\ (\sigma_{\uparrow N}, \sigma_{\uparrow N}) \in G}{\Gamma, \Theta \vdash_{/F}\ DynCom\ c\ \textbf{sat}\ [p, R, G, q, a]}$$

[Par]

$$\frac{\begin{array}{l}\forall i \le n.\ \Gamma, \Theta \vdash_{/F}\ C_i\ \textbf{sat}\ [p_i, R_i, G_i, q_i, a_i] \\ \forall i \le n.\ R \cup (\bigcup j \in \{j.\ j \le n \wedge j \neq i\}.\ G_i) \subseteq R_i \\ (\bigcup j \le n.\ G_j) \subseteq G \quad p \subseteq (\bigcap i < n.\ p_n) \\ (\bigcap j \le n.\ q_j) \subseteq q \quad (\bigcup j \le n.\ a_j) \subseteq a\end{array}}{\Gamma, \Theta \vdash_{/F}\ [C_0 || \ldots || C_n]\ \textbf{SAT}\ [p, R, G, q, a]}$$

4. Rely-Guarantee for CSimpl

Soundness of the Proof System

Thanks!