

Static Analysis via Abstract Interpretation of the Happens-Before Memory Model

Pietro Ferrara

Ecole Polytechnique, Paris
F-91128 Palaiseau (France)
Università Ca' Foscari di Venezia
I-30170 Venezia (Italy)
Pietro.Ferrara@polytechnique.edu

April 10, 2008

- ▶ Multicore architectures
 - ▶ many parallel tasks
- ▶ More difficult than sequential programs [SUT05]
 - ▶ interleaving of the executions of different threads
 - ▶ implicit communications through the shared memory
 - ▶ specification of the memory model [LEE06]

Memory Model, what is it?

- ▶ Defines which multithreaded executions are legal
- ▶ Two opposite needs:
 - ▶ restrict the non-deterministic behaviors
 - ▶ allow as many compiler optimizations as possible
- ▶ e.g. sequential consistency:
 - ▶ “...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program” [LAM79]
- ▶ too much restrictive in practice, it does not allow the most part of optimizations

An example

Thread 1	Thread 2
<code>i = 1;</code>	<code>if(j == 1&& i == 0)</code>
<code>j = 1;</code>	<code>throw new Exception();</code>

- ▶ Supposing that at the beginning $i = 0, j = 0$
- ▶ May the exception be thrown?
 - ▶ sequential consistency: NO!
 - ▶ practical needs: YES!
 - ▶ swap of independent statements

The Happens-Before Memory Model

- ▶ Program order:
 - ▶ intra-thread order of statements
- ▶ Synchronizes-with relation: involve two **synchronized** actions.
E.g. the acquisition and the release of a monitor
- ▶ Happens-before order [LAM78]: a_1 happens-before a_2 if
 - ▶ a_1 appears before a_2 in the program order
 - ▶ a_2 synchronizes-with a_1
 - ▶ you can reach a_2 by following happens-before edges starting from a_1

Synchronizes-with

- ▶ Many different ways of synchronization:
 - ▶ mutual exclusion on monitors
 - ▶ wait and notify on objects
 - ▶ the first action of a thread synchronizes-with the statement that launched it
 - ▶ ...
- ▶ We are generic w.r.t. the programming language
- ▶ We focus on
 - ▶ mutual exclusion
 - ▶ launch of a thread
- ▶ Other synchronizations... future work!

Already an approximation

- ▶ The memory model defines which behaviors are allowed
- ▶ Approximation of all the possible real multithread architectures
- ▶ Different
 - ▶ The implementation of a system defines which value is seen at a given point of the computation
 - ▶ The memory model defines which values may be seen between ALL the ones produced by other threads

Our contribution

- ▶ A **static analysis**
- ▶ On all the multithreaded programs
- ▶ Sound w.r.t. the happens-before memory model
- ▶ Based on the **abstract interpretation** theory

Abstract interpretation

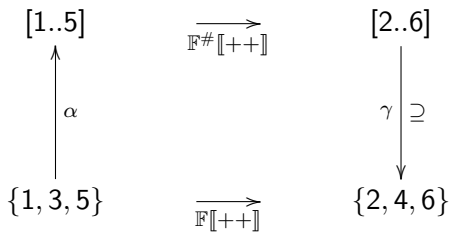
- ▶ Static analysis
- ▶ Mathematical theory to define and soundly approximate the semantics of a program [CC77,CC79]
- ▶ First step: concrete domain and semantics - not computable!
- ▶ Then: abstract domain and semantics - computable!
- ▶ Finally: correctness

- ▶ Concrete domain: $\langle A, \sqsubseteq_C, \perp_C, \top_C, \sqcup_C, \sqcap_C \rangle$
- ▶ Usually the concrete domain is composed by powerset (e.g. all the possible states of a program in a given point) - $\langle \wp(A), \subseteq, \emptyset, A, \cup, \cap \rangle$
- ▶ and the abstract domain approximates it with an unique element - $\langle \bar{A}, \sqsubseteq_A, \perp_A, \top_A, \sqcup_A, \sqcap_A \rangle$
- ▶ Abstraction α and concretization γ functions
- ▶ The abstraction is correct iff it forms a Galois connection, i.e.
 - ▶ $\forall S \in \wp(A) : S \subseteq \gamma(\alpha(S))$
 - ▶ $\forall \bar{s} \in \bar{A} : \bar{s} \sqsubseteq_A \alpha(\gamma(\bar{s}))$

- ▶ Usually defined as the computation of a fixpoint
- ▶ Concrete: all the possible executions
- ▶ Abstract: approximates all the concrete executions with an unique trace
- ▶ Relies on a transfer function (the semantics of statements)
- ▶ The abstract transfer function has to soundly approximate the concrete one:

$$\forall S \in \wp(A) : \mathbb{F}[[S]] \subseteq \gamma(\mathbb{F}^\#[[\alpha(S)]])$$

An example



Trace Semantics

- ▶ Often used in the abstract interpretation framework
- ▶ Semantics of a system as sequence (e.g. of states)
- ▶ Given a set S , $S^{\vec{\tau}}$ is the set of finite traces composed by elements of this set
 - ▶ $\sigma_0, \dots, \sigma_i \in S$
 - ▶ $\sigma_0 \rightarrow \dots \rightarrow \sigma_i \in S^{\vec{\tau}}$

Single-thread Domain

- ▶ Generic w.r.t. the programming language ...
- ▶ ... we do not formally define in the details the concrete domain!
- ▶ Some functions provide us the essential information
 - ▶ $shared : St \mapsto Sh$
 - ▶ $action : St \mapsto \perp_a \cup (\{r, w\} \times Loc \times Val)$
 - ▶ $synchronised : St \mapsto \wp(Sync)$
- ▶ Parameterized also on the intra-thread transfer function $\overset{\circ}{\rightarrow}$
[FER08] defines a concrete domain and the semantics of the intra-thread transfer function on a subset of the Java bytecode language

- ▶ For each thread we collect the trace representing its execution

$$\Psi : TId \rightarrow St^{\vec{t}}$$

$$\Omega : TId \rightarrow ((TId \times Integer) \cup \perp_{\Omega})$$

- ▶ $t_1 \mapsto (t_2, 5)$ means that thread t_1 has been launched by thread t_2 at the instruction at program counter 5

Intra-thread semantics

- ▶ *visible* function: return all the values written in parallel by other threads that may be seen following the HB memory model
- ▶ *step* function: defines a single step in the computation. Read from the shared memory one of the visible values \rightarrow nondeterministic choices

Definition (\mathbb{S}°)

$$\mathbb{S}^\circ : \Psi \times \Omega \times TId \mapsto \wp(St^{\vec{T}})$$

$$\mathbb{S}^\circ \llbracket f, r, t \rrbracket = \text{lfp}_{\emptyset}^{\subseteq} \lambda T. \{ \sigma_0 \} \cup \{ \sigma_0 \rightarrow \dots \rightarrow \sigma_{i-1} \rightarrow \sigma_i : \\ \sigma_0 \rightarrow \dots \rightarrow \sigma_{i-1} \in T \wedge \sigma_i \in \text{step}(t, f, r) \}$$

- ▶ We do not enter in the formal details of the definition of *visible*
- ▶ The paper presents it in the details
- ▶ It uses
 - ▶ the Ω element to check the values written by a thread before launching the current one
 - ▶ the *synchronised* function in order to support mutual exclusion

Multithread semantics

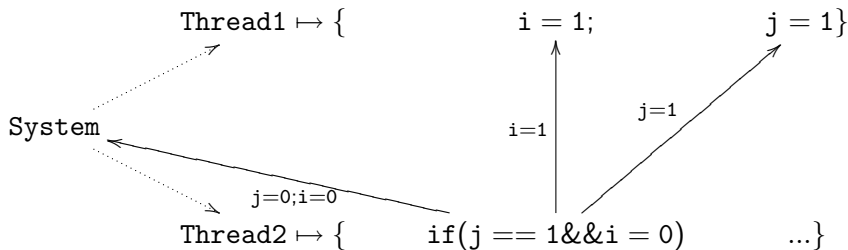
- ▶ Each time we compute the single thread semantics of a thread it may expose new values to other threads
- ▶ Iterates the single-thread semantics until a fixpoint is reached

Definition (S^{\parallel})

$$S^{\parallel} : \Psi \times \Omega \mapsto \wp(\Psi \times \Omega)$$

$$S^{\parallel} \llbracket f_0, r_0 \rrbracket = \text{lfp}_{\emptyset}^{\subseteq} \lambda \Phi. \{(f_0, r_0)\} \cup \{(f_i, r) : \exists (f_{i-1}, r) \in \Phi : \\ \forall t \in \text{dom}(f_{i-1}) : f_i(t) \in S^{\circ} \llbracket f_{i-1}, r, t \rrbracket, \\ f_i(t) = \sigma_0 \rightarrow \dots \rightarrow \sigma_i, \sigma_i \in \text{St}_{\rightarrow}\}$$

An example



$visible(j) = \{0, 1\}$

$visible(i) = \{0, 1\}$

- ▶ Exactly the same components of the concrete domain, but dealing with abstract elements
- ▶ Functions:
 - ▶ $shared^\# : \overline{St} \mapsto \overline{Sh}$
 - ▶ $action^\# : \overline{St} \mapsto \overline{\perp}_a \cup (\{\mathbf{r}, \mathbf{w}\} \times \overline{Loc} \times \overline{Val})$
 - ▶ $synchronised^\# : \overline{St} \mapsto \wp(\overline{Sync})$

[FER08] abstracts the concrete domain and intra-thread semantics

- ▶ Multithreaded state:

$$\begin{aligned}\overline{\Psi} : \overline{Tld} &\rightarrow \overline{St}^\dagger \\ \overline{\Omega} : \overline{Tld} &\rightarrow ((\overline{Tld} \times \overline{Integer}) \cup \overline{\perp}_\Omega)\end{aligned}$$

Intra-thread semantics

- ▶ $visible^\#$ function: straight abstraction of the concrete one!
- ▶ $step^\#$ function: defines a single step in the computation.
Makes the upper bound between all the visible values \rightarrow deterministic choice!

Definition (Single-thread semantics \bar{S}°)

$$\begin{aligned} \bar{S}^\circ & : (\bar{\Psi} \times \bar{\Omega} \times \bar{TId}) \mapsto \bar{St}^\ddagger \\ \bar{S}^\circ \llbracket \bar{f}, \bar{r}, \bar{t} \rrbracket & = lfp_{\emptyset}^{\sqsubseteq} \lambda \bar{\tau}. \{ \bar{\sigma}_0 \} \sqcup_{\bar{\tau}} \{ \bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_{i-1} \rightarrow \bar{\sigma}_i : \\ & \quad \bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_{i-1} = \bar{\tau} \wedge \bar{\sigma}_i = step^\#(\bar{t}, \bar{f}, \bar{r}) \} \end{aligned}$$

Multithread semantics

- ▶ Each time we compute the single thread semantics of a thread it may expose new values to other threads
- ▶ Iterates the single-thread semantics until a fixpoint is reached
- ▶ ... as in the concrete semantics!

Definition (Multithreaded semantics \bar{S}^{\parallel})

$$\bar{S}^{\parallel} : \bar{\Psi} \times \bar{\Omega} \mapsto \bar{\Psi} \times \bar{\Omega}$$

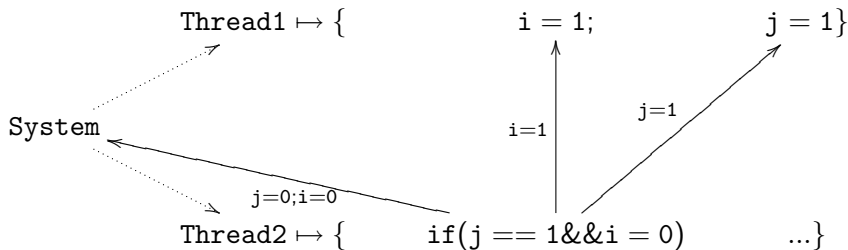
$$\bar{S}^{\parallel} \llbracket \bar{f}_0, \bar{r}_0 \rrbracket = \text{Ifp}_{\emptyset}^{\sqsubseteq} \lambda(\bar{f}, \bar{r}). \{(\bar{f}_0, \bar{r}_0)\} \sqcup_f \{(\bar{f}_i, \bar{r}) : \forall \bar{t} \in \text{dom}(\bar{f}) : \bar{f}_i(\bar{t}) = \bar{S}^{\circ} \llbracket \bar{f}, \bar{t} \rrbracket\}$$

We formally proved the correctness of our analysis, i.e. :

Theorem (Soundness of \bar{S}^{\parallel})

$$\forall \bar{f} \in \bar{\Psi}_{pre} : \alpha_f(S^{\parallel})[\bar{f}] \sqsubseteq_f \bar{S}^{\parallel}[\bar{f}].$$

An example



$$\text{visible}^\#(j) = [0..0] \sqcup [1..1] = [0..1]$$

$$\text{visible}^\#(i) = [0..0] \sqcup [1..1] = [0..1]$$

Related works

- ▶ Many papers have been focused on the analysis of specific properties like data race condition [RIN01]. Usually these approaches suppose that the executions are sequentially consistent
- ▶ Other papers define memory models [MAN05,CEN07,SAR07] but they do not propose any static analysis
- ▶ Two model checkers [ROY02,HUY06] sound w.r.t. a memory model have been proposed in the last years. They are afflicted by the static explosion problem, and they seem not in position to scale up
- ▶ Our paper does not present any experimental result. We have implemented a similar approach in another work [FER08] and the first results are encouraging (about 1'10" in order to analyze 24 threads and more than 2.000 statements)

Conclusion

- ▶ We formalize in fixpoint form the happens-before memory model
 - ▶ generic w.r.t. the programming language
 - ▶ supports mutual exclusion
- ▶ We abstract it
 - ▶ The main characteristic components of the memory model are straightly abstracted!
 - ▶ For the next memory model, define the concrete semantics and obtain the abstract one!
- ▶ We prove formally the soundness of our abstraction
- ▶ As far as we know, it is the first static analysis sound w.r.t. the happens-before memory model

[FER08] Pietro Ferrara. A fast and precise analysis for data race detection. In Elsevier, editor, Proceedings of Bytecode 08, volume ENTCS, 2008.

- [CC77]** Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL 77, pages 238-252, Los Angeles, California, 1977. ACM Press.
- [CC79]** Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In POPL 79, pages 269-282, San Antonio, Texas, 1979. ACM Press.

Bibliography - Memory Models and Concurrency

- [LAM79]** Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. In IEEE Trans. Computers, volume 28, pages 690691, 1979.
- [LAM78]** Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In Commun. ACM, volume 21, pages 558565, New York, NY, USA, 1978. ACM Press.
- [MAN05]** J. Manson, W. Pugh, and S. V. Adve. The Java memory model. In ACM Press, editor, Proceedings of POPL 05, 2005.
- [LEE06]** Edward A. Lee. The problem with threads. In IEEE Computer, volume 39, pages 3342, Los Alamitos, CA, USA, May 2006. IEEE Computer Society Press.
- [SUT05]** Herb Sutter and James Larus. Software and the concurrency revolution. In ACM Queue, Vol. 3, No. 7, pages 5462, September 2005.

Bibliography - Related Works

[RIN01] Martin C. Rinard. Analysis of multithreaded programs. In SAS 01, pages 119, London, UK, 2001. Springer-Verlag.

[CEN07] Pietro Cenciarelli, Alexander Knapp, and Eleonora Sibilio. The java memory model: Operationally, denotationally, axiomatically. In Springer Verlag, editor, Proceeding of ESOP 07, number 4421 in Lecture Notes in Computer Science, pages 331346, 2007.

[SAR07] Vijay Saraswat, Radha Jagadeesan, Maged Michael, and Christoph von Praun. A theory of memory models. To appear (<http://www.saraswat.org/raofull.pdf>), 2006.

[ROY02] Abhik Roychoudhury and Tulika Mitra. Specifying multithreaded java semantics for program verification. In ACM Press, editor, Proceedings of ICSE, May 2002.

[HUY06] Thuan Quang Huynh and Abhik Roychoudhury. A memory model sensitive checker for c#. In Springer-Verlag, editor, FM, volume 4085 of Lecture Notes in Computer Science, 2006.

Definition (*step function*)

step : $TId \times \Psi \times \Omega \mapsto \wp(St)$

step $(t, f, s) = \{\sigma\}$ where $f(t) = \sigma_0 \rightarrow \dots \rightarrow \sigma_i$ and

(1) $\sigma_i \overset{\circ}{\rightarrow} \sigma$ if $\pi_1(action(\sigma_i)) \neq r$

(2a) $\sigma_i \overset{\circ}{\rightarrow} \sigma \vee$ if $\pi_1(action(\sigma_i)) = r$

(2b) $\exists v \in visible(t, \pi_2(action(\sigma_i)), synchronised(\sigma_i), f, s(t)) :$
 $\sigma' = set_shared(\sigma_i, shared(\sigma_i)[l \mapsto v]), \sigma' \overset{\circ}{\rightarrow} \sigma$

Definition (*visible* function)

$visible : TId \times Loc \times \wp(Sync) \times \Psi \times ((TId \times Integer) \cup \perp_{\Omega}) \mapsto \wp(Val)$

$visible(t, l, S, f, (t', i')) =$

(1) $= project(l, suffix(f(t'), i'), S) \cup$

(2) $\{v : v \in project(l, f(t''), S) : t'' \in dom(f) \setminus \{t, t'\}\}$

Definition (*suffix* function)

$$\text{suffix} : St^{\vec{\tau}} \times Integer \mapsto St^{\vec{\tau}}$$

$$\text{suffix} (\sigma_0 \rightarrow \cdots \rightarrow \sigma_j, i) = \begin{cases} \sigma_i \rightarrow \cdots \rightarrow \sigma_j & \text{if } i \geq 0 \wedge i < j \\ \epsilon & \text{if } i = j \end{cases}$$

Definition (*project* function)

$project : Loc \times St^{\vec{t}} \times \wp(Sync) \mapsto \wp(Val)$

$project (l, \sigma_0 \rightarrow \dots \rightarrow \sigma_i, S) = \{v : \exists j \in [0..i] :$

$action(\sigma_j) = (w, l, v) \wedge not_synchronised(\sigma_j \rightarrow \dots \rightarrow \sigma_i, S)\}$

Definition (*not_synchronised* function)

$not_synchronised : St^{\vec{\tau}} \times \wp(Sync) \mapsto \{\text{true}, \text{false}\}$

$not_synchronised(\sigma_0 \rightarrow \dots \rightarrow \sigma_i, S) = \text{true}$ if and only if

(1) $S \cap synchronised(\sigma_0) = \emptyset \vee$

(2) $\nexists \sigma_j \in cut(\sigma_0 \rightarrow \dots \rightarrow \sigma_i, S) : action(\sigma_j) = (\mathbf{w}, l, \mathbf{v}),$
 $action(\sigma_0) = (\mathbf{w}, l_0, \mathbf{v}_0), l = l_0$

Definition (*cut* function)

$$\textit{cut} : St^{\vec{\tau}} \times \wp(\textit{Sync}) \mapsto St^{\vec{\tau}}$$

$$\textit{cut} (\sigma_0 \rightarrow \dots \rightarrow \sigma_i, S) = \begin{cases} \epsilon & \text{if } \textit{synchronised}(\sigma_0) \cap S = \emptyset \\ \sigma_0 \rightarrow \textit{cut}(\sigma_1 \rightarrow \dots \rightarrow \sigma_i, S) & \text{otherwise} \end{cases}$$

Definition (*step*[#] function)

$$\text{step}^\# : \overline{TId} \times \overline{\Psi} \times \overline{\Omega} \mapsto \overline{St}$$

$$\text{step}^\# (\bar{t}, \bar{f}, \bar{s}) = \sigma \text{ where } \bar{f}(\bar{t}) = \bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_i \text{ and}$$

$$\bar{\sigma}_i \overset{\circ}{\rightarrow} \# \bar{\sigma} \quad \text{if } \pi_1(\text{action}^\#(\bar{\sigma}_i)) \neq r$$

$$\bar{\sigma}'_i \overset{\circ}{\rightarrow} \# \bar{\sigma} : \quad \text{if } \pi_1(\text{action}^\#(\bar{\sigma}_i)) = r$$

$$\overline{V} = \text{visible}^\#(\bar{t}, \pi_2(\text{action}^\#(\bar{\sigma}_i)), \text{synchronised}^\#(\bar{\sigma}_i), \bar{f}, \bar{s}(\bar{t}))$$

$$\bar{v} = \bigsqcup_{\bar{v}' \in \overline{V}} \bar{v}'$$

$$\overline{sh} = \text{shared}^\#(\bar{\sigma}_i), \overline{sh}' = \overline{sh}[\bar{l} \mapsto \bar{v} \sqcup_{Val} \overline{sh}(\bar{l})]$$

$$\bar{\sigma}'_i = \text{set_shared}^\#(\bar{\sigma}, \overline{sh}')$$

Definition (*visible*[#] function)

$$\begin{aligned} \text{visible}^\# &: \overline{TId} \times \overline{Loc} \times \wp(\overline{Sync}) \times \overline{\Psi} \times (\overline{TId} \times \text{Integer}) \mapsto \wp(\overline{Val}) \\ \text{visible}^\# &(\bar{t}, \bar{l}, \bar{S}, \bar{f}, (\bar{t}', i')) = \\ &= \text{project}^\#(\bar{l}, \text{suffix}^\#(\bar{f}(\bar{t}'), i'), \bar{S}) \cup \\ &\{\bar{v} : \bar{v} \in \text{project}^\#(\bar{l}, \bar{f}(\bar{t}''), \bar{S}) : \bar{t}'' \in \text{dom}(\bar{f}) \setminus \{\bar{t}, \bar{t}'\}\} \end{aligned}$$

Definition (*suffix*[#] function)

$$\begin{aligned} \text{suffix}^\# &: \overline{\text{St}}^{\vec{\tau}} \times \text{Integer} \mapsto \overline{\text{St}}^{\vec{\tau}} \\ \text{suffix}^\# (\bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_j, i) &= \begin{cases} \bar{\sigma}_i \rightarrow \dots \rightarrow \bar{\sigma}_j & \text{if } i \geq 0 \wedge i < j \\ \bar{\epsilon} & \text{if } i = j \end{cases} \end{aligned}$$

Definition (*project*[#] function)

$$project^{\#} : \overline{Loc} \times \overline{St}^{\vec{\dagger}} \times \wp(\overline{Sync}) \mapsto \wp(\overline{Val})$$

$$project^{\#}(\bar{l}, \bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_i, \bar{S}) = \{\bar{v} : \exists j \in [0..i] :$$

$$action^{\#}(\bar{\sigma}_j) = (\bar{w}, \bar{l}, \bar{v}) \wedge not_synchronised^{\#}(\bar{\sigma}_j \rightarrow \dots \rightarrow \bar{\sigma}_i, \bar{S})\}$$

Definition (*not_synchronised*[#] function)

$$\begin{aligned} \text{not_synchronised}^\# &: \overline{St}^\dagger \times \wp(\overline{Sync}) \mapsto \{\text{true}, \text{false}\} \\ \text{not_synchronised}^\#(\bar{\sigma}_0 \rightarrow \dots \rightarrow \bar{\sigma}_i, \bar{S}) &= \text{true if and only if} \\ &\bar{S} \cap \text{synchronised}^\#(\bar{\sigma}_0) = \emptyset \vee \\ &\nexists \bar{\sigma}_j \in \text{cut}^\#(\bar{\sigma}_0 \rightarrow \dots \bar{\sigma}_i, \bar{S}) : \\ &\text{action}^\#(\bar{\sigma}_{j-1}) = (\mathbf{w}, \bar{l}, \bar{v}), \text{action}^\#(\bar{\sigma}_0) = (\mathbf{w}, \bar{l}_0, \bar{v}_0), \bar{l} = \bar{l}_0 \end{aligned}$$

Definition ($cut^\#$ function)

$$\begin{aligned} cut^\# &: \overline{St}^{\vec{\tau}} \times \wp(\overline{Sync}) \mapsto \overline{St}^{\vec{\tau}} \\ cut^\#(\overline{\sigma}_0 \rightarrow \cdots \rightarrow \overline{\sigma}_i, \overline{S}) &= \\ &= \begin{cases} \epsilon & \text{if } \text{synchronised}^\#(\overline{\sigma}_0) \cap \overline{S} = \emptyset \\ \overline{\sigma}_0 \rightarrow cut^\#(\overline{\sigma}_1 \rightarrow \cdots \rightarrow \overline{\sigma}_i, \overline{S}) & \text{otherwise} \end{cases} \end{aligned}$$