

Complexity Results for May-Happen-in-Parallel Analysis

Jonathan K. Lee, Jens Palsberg, and Rupak Majumdar

UCLA, University of California, Los Angeles

Abstract. For concurrent and parallel languages, may-happen-in-parallel (MHP) analysis is useful as a basis for tools such as data race detectors. While many approximate static MHP analyses exist, researchers have published only a few papers on decidability results for MHP analysis. We study MHP analysis for a model of X10, a parallel language with async-finish parallelism. For programs with procedures, we show that the MHP decision problem is decidable in linear time, and hence the set of pairs of actions that may happen in parallel can be computed in cubic time. For programs without procedures, we present a practical recursive decision procedure that does multiple-query MHP analysis in cubic time. Our results indicate that MHP analysis is tractable for a standard storeless abstraction of X10 programs.

1 Introduction

For concurrent and parallel languages, may-happen-in-parallel (MHP) analysis is useful as a basis for tools such as data race detectors [6, 12]. A may-happen-in-parallel analysis determines whether a pair of statements may happen in parallel during some execution of the program.

In general, MHP analysis is undecidable for parallel recursive programs even when the store is abstracted [15], but can be decidable when language features are restricted (e.g., it is easily decidable for multithreaded non-recursive programs without thread creation). For example, Taylor [16] studied MHP analysis for concurrent programs that use rendezvous as their synchronization mechanism. For a set of tasks that each contain only straight-line code, Taylor showed that the MHP analysis problem is NP-complete, even if the set of possible rendezvous is known. Kahlon [8] studied MHP analysis for concurrent programs that use locks as their synchronization mechanism. For programs with non-nested lock usage in which every lock chain is bounded, Kahlon showed that the MHP analysis problem is decidable, although he did not give a more specific upper bound. So far, decidability results have been mostly for theoretical interest. Instead, most previous papers on practical MHP analysis present static analyses that give conservative, approximate answers [7, 11, 13, 14, 10, 3, 1, 9].

In this paper, we study MHP analysis for a model of X10 [5], a parallel language with async-finish parallelism. The async statement is a lightweight notation for spawning threads, while a finish statement `finish s` waits for termination of all async statement bodies started while executing `s`. X10 programs

use finish statements as their synchronization mechanism; the finish statement is entirely different from rendezvous and locks. Our main result is a *linear time* algorithm for MHP analysis.

As with any MHP analysis, our model of X10 abstracts away the store. The result is an MHP analysis problem that is all about control flow. Our starting point is Featherweight X10 [9], which is a core calculus for async-finish parallelism and essentially a subset of X10. We give a store-less abstract semantics of Featherweight X10 and define the MHP analysis problem in terms of that.

We distinguish between the MHP decision problem that concentrates on a single pair of statements, and multiple-query MHP analysis that provides the answer for all pairs of statements.

For programs with (potentially recursive) procedures, we show that single-query MHP analysis is decidable in linear time, and that multiple-query MHP analysis is thus computable in cubic time. Our proof uses a reduction from X10 programs to constrained dynamic pushdown networks (CPDNs) [4]. We give a careful complexity analysis of a known decision procedure for CPDNs [4] for when it is applied to the CPDNs produced by our reduction.

For programs without procedures, we present a practical recursive decision procedure that does multiple-query MHP analysis in cubic time. For programs with procedures, we can extend our recursive decision procedure to give conservative, approximate answers, via the constraint technique of Lee and Palsberg [9]. Compared to the decision procedure based on CPDNs, our recursive decision procedure has low overhead and is straightforward to implement.

In the following section we recall Featherweight X10 and give it an abstract semantics, and in Section 3 we define the MHP analysis problem. In Section 4 we show our reduction from the MHP analysis problem to CPDNs that leads to an efficient decision procedure. In Section 5 we present a type system that characterizes the MHP analysis problem for programs without procedures, and in Section 6 we prove it correct. A straightforward recursive procedure performs type inference and thereby solves the multi-query MHP analysis problem. Our results demonstrate a *tractable* MHP analysis for a *practical* parallel programming language. We have omitted some proofs; they are given in the full version of the paper which is available from second author's webpage.

2 Featherweight X10

We now recall Featherweight X10 [9], in a simplified version that is what we need for giving a store-less abstract semantics. In contrast to the paper that introduced Featherweight X10, we give a semantics based on evaluation contexts.

Syntax. Figure 1 gives the syntax of statements, contexts, parallel statements, and redexes, as well as a function for plugging a statement into a context. In the production for *Statement*, $s ; s$ denotes statement sequence, *loop* s executes s zero, one, or more times, *async* s spawns off s in a separate thread, *finish* s waits for termination of all *async* statement bodies started while executing s , a^l is a primitive statement with label l , and *skip* is the empty statement.

$$\begin{aligned}
(\text{Statement}) \quad s &::= s ; s \mid \text{loop } s \mid \text{async } s \mid \text{finish } s \mid a^l \\
&\mid \text{skip} \\
(\text{Context}) \quad C &::= C ; s \mid P ; C \mid \text{async } C \mid \text{finish } C \mid \square \\
(\text{ParStatement}) \quad P &::= P ; P \mid \text{async } s \\
(\text{Redex}) \quad R &::= \text{skip} ; s \mid P ; \text{skip} \mid \text{loop } s \mid \text{async } \text{skip} \\
&\mid \text{finish } \text{skip} \mid a^l
\end{aligned}$$

$$\begin{aligned}
-[_] &: \text{Context} \times \text{Statement} \rightarrow \text{Statement} \\
(C ; s)[s'] &= (C[s']) ; s \\
(P ; C)[s'] &= P ; (C[s']) \\
(\text{async } C)[s'] &= (\text{async } C[s']) \\
(\text{finish } C)[s'] &= (\text{finish } C[s']) \\
(\square)[s'] &= s'
\end{aligned}$$

Fig. 1. Syntax of Featherweight X10

Notice that every parstatement is a statement and that every redex is a statement.

The following theorem characterizes statements in terms of contexts and redexes, and is proved by induction on the structure of s .

Theorem 1. (Statement Characterization) *For every statement s , either $s = \text{skip}$, or there exists a context C and a redex R such that $s = C[R]$.*

Proof. We proceed by induction on s . We have six cases.

- $s = s_1 ; s_2$. From the induction hypothesis we have that either $s_1 = \text{skip}$ or there exists C_1 and R_1 such that $s_1 = C_1[R_1]$. Let us consider each case in turn.
 - $s_1 = \text{skip}$. In this case we have $s = \text{skip} ; s_2 = \square[\text{skip} ; s_2]$.
 - There exists C_1 and R_1 such that $s_1 = C_1[R_1]$. In this case we have $s = (C_1[R_1]) ; s_2 = (C_1 ; s_2)[R_1]$.
- $s = \text{loop } s_1$. We have $s = (\square)[\text{loop } s_1]$.
- $s = \text{async } s_1$. From the induction hypothesis we have that either $s_1 = \text{skip}$ or there exists C_1 and R_1 such that $s_1 = C_1[R_1]$. Let us consider each case in turn.
 - $s_1 = \text{skip}$. In this case we have $s = \text{async } \text{skip} = \square[\text{async } \text{skip}]$.
 - There exists C_1 and R_1 such that $s_1 = C_1[R_1]$. In this case we have $s = \text{async } (C_1[R_1]) = (\text{async } C_1)[R_1]$.
- $s = \text{finish } s_1$. From the induction hypothesis we have that either $s_1 = \text{skip}$ or there exists C_1 and R_1 such that $s_1 = C_1[R_1]$. Let us consider each case in turn.
 - $s_1 = \text{skip}$. In this case we have $s = \text{finish } \text{skip} = \square[\text{finish } \text{skip}]$.

- There exists C_1 and R_1 such that $s_1 = C_1[R_1]$. In this case we have
 - $s = \text{finish } (C_1[R_1]) = (\text{finish } C_1)[R_1]$.
 - $s = a^l$. We have $s = (\square)[a^l]$.
 - $s = \text{skip}$. In this case the result is immediate.

Abstract Semantics. We will now define a small-step abstract store-less operational semantics. First we define a relation $\rightarrow \subseteq \text{Redex} \times \text{Statement}$:

$$\text{skip} ; s \rightarrow s \quad (1)$$

$$P ; \text{skip} \rightarrow P \quad (2)$$

$$\text{loop } s \rightarrow \text{skip} \quad (3)$$

$$\text{loop } s \rightarrow s ; \text{loop } s \quad (4)$$

$$\text{async skip} \rightarrow \text{skip} \quad (5)$$

$$\text{finish skip} \rightarrow \text{skip} \quad (6)$$

$$a^l \rightarrow \text{skip} \quad (7)$$

Notice that for every redex R there exists s such that $R \rightarrow s$. Next we define a relation $\mapsto \subseteq \text{Statement} \times \text{Statement}$:

$$C[R] \mapsto C[s] \iff R \rightarrow s$$

3 The May Happen in Parallel Problem

In this section we define the May Happen in Parallel problem.

CAN BOTH EXECUTE

Instance: (s, l_1, l_2) where s is a statement and l_1, l_2 are labels.

Problem: Does there exist C_1, C_2 such that $C_1 \neq C_2$ and $s = C_1[a^{l_1}] = C_2[a^{l_2}]$?

We will use $\text{CBE}(s, l_1, l_2)$ to denote the answer to the CAN BOTH EXECUTE problem for the instance (s, l_1, l_2) . We define

$$\text{CBE}(s) = \{ (l_1, l_2) \mid \text{CBE}(s, l_1, l_2) \}$$

The decision version of the MHP problem is defined as follows.

MAY HAPPEN IN PARALLEL (DECISION PROBLEM)

Instance: (s, l_1, l_2) where s is a statement and l_1, l_2 are labels.

Problem: Does there exist s' such that $s \mapsto^* s'$ and $\text{CBE}(s', l_1, l_2)$?

We will use $\text{MHP}_{\text{sem}}(s, l_1, l_2)$ to denote the answer to the MAY HAPPEN IN PARALLEL problem for the instance (s, l_1, l_2) . We use the subscript *sem* to emphasize that the definition is semantics based. We define

$$\text{MHP}_{\text{sem}}(s) = \{ (l_1, l_2) \mid \text{MHP}_{\text{sem}}(s, l_1, l_2) \}$$

Notice that $\text{MHP}_{\text{sem}}(s) = \bigcup_{s': s \mapsto^* s'} \text{CBE}(s')$. The computation version of the MHP problem (called the *MHP computation problem* in the following) takes as input a statement s and outputs $\text{MHP}_{\text{sem}}(s)$.

4 From X10 to CDPN

We now demonstrate a general algorithm for the MHP problem for programs even in the presence of recursive procedures. We omit the straightforward details of extending the syntax and semantics in Section 2 with parameterless recursive procedures and procedure call.

Our main technical construction is a reduction from the MHP problem to constrained dynamic pushdown networks (CPDNs) [4], an infinite model of computation with nice decidability properties. Informally, CPDNs model collections of sequential pushdown processes running in parallel, where each process can “spawn” a new process or, under some conditions, observe the state of its children. We start with some preliminary definitions, following the presentation in [4].

Let Σ be an alphabet, and let $\rho \subseteq \Sigma \times \Sigma$ be a binary relation on Σ . A set $S \subseteq \Sigma$ is ρ -stable iff for each $s \in S$ and for each $t \in \Sigma$, if $(s, t) \in \rho$ then t is also in S . A ρ -stable regular expression over Σ is defined inductively by the grammar:

$$e ::= S \mid e \cdot e \mid e^*$$

where S is a ρ -stable set. We derive a ρ -stable regular language from a ρ -stable regular expression in the obvious way.

A *constrained dynamic pushdown network* (CDPN) [4] (A, P, Γ, Δ) consists of a finite set A of *actions*, a finite set P of *control locations*, a finite alphabet Γ of stack symbols (disjoint from P), and a finite set Δ of *transitions* of the following forms: (a) $\phi : p\gamma \xrightarrow{a} p_1w_1$ or (b) $\phi : p\gamma \xrightarrow{a} p_1w_1 \triangleright p_2w_2$, where $p, p_1, p_2 \in P$, $\gamma \in \Gamma$, $w_1, w_2 \in \Gamma^*$, and ϕ is a ρ_Δ -stable regular expression over P with

$$\rho_\Delta = \{(p, p') \in P \times P \mid \text{there is a rule } \psi : p\gamma \xrightarrow{a} p'w \text{ or } \psi : p\gamma \xrightarrow{a} p'w \triangleright p''w' \text{ in } \Delta\}$$

In the following, we identify the regular expression ϕ with the regular language represented by the regular expression.

Intuitively, a configuration of a CDPN is a tree where each node is marked with the configuration of a pushdown process, and the children of a node are its children ordered by age (the more recently spawned child to the right). Formally, given a set $X = \{x_1, \dots, x_n\}$ of variables, define the set $\mathcal{T}[X]$ of M -terms over $X \cup P \cup \Gamma$ as the smallest set satisfying: $X \subseteq \mathcal{T}[X]$; if $t \in \mathcal{T}[X]$ and $\gamma \in \Gamma$, then $\gamma(t) \in \mathcal{T}[X]$; and for each $n \geq 0$, if $t_1, \dots, t_n \in \mathcal{T}[X]$ and $p \in P$, then $p(t_1, \dots, t_n) \in \mathcal{T}[X]$. An M -configuration is a term in $\mathcal{T}[X]$ (i.e., a term without free variables, also called *ground terms*). A ground term $\gamma_m \dots \gamma_1 p(t_1, \dots, t_n)$ represents a configuration of a CDPN in which the common ancestor to all processes is in control location p and has $\gamma_1 \dots \gamma_m$ in the stack (with γ_1 on the top), and this ancestor has n (spawned) children, the i th child, along with all its descendants given by t_i for $i = 1, \dots, n$.

The semantics of a CDPN is given as a binary transition relation \rightarrow_M between M -configurations. We define a context C as a term with one free variable, which moreover appears at most once in the term. If t is a ground term, then $C[t]$ is the ground term obtained by substituting the free variable with t . Given a

configuration t of one of the forms $\gamma_m \dots \gamma_1 p(t_1, \dots, t_n)$ or $\gamma_m \dots \gamma_1 p$, we define $root(t)$ to be the control location p of the topmost process in t . We define \rightarrow_M as the smallest relation such that the following hold: (a) if $(\phi : p\gamma \xrightarrow{a} p_1 w_1) \in \Delta$ and $root(t_1) \dots root(t_n) \in \phi$ then $C[\gamma p(t_1, \dots, t_n)] \rightarrow_M C[w_1^R p_1(t_1, \dots, t_n)]$; and (b) if $(\phi : p\gamma \xrightarrow{a} p_1 w_1 \triangleright p_2 w_2) \in \Delta$ and $root(t_1) \dots root(t_n) \in \phi$ then $C[\gamma p(t_1, \dots, t_n)] \rightarrow_M C[w_1^R p_1(t_1, \dots, t_n, w_2^R p_2)]$.

Given the transition relation \rightarrow_M , we define pre, pre^* in the usual way.

From programs to CDPNs. We now give a translation from programs in our syntax to CDPNs. Our translation starts with a control-flow graph (CFG) representation of a program, in which each procedure f is represented as a labeled, directed graph $G_f = (V_f, E_f, \text{entry}_f, \text{exit}_f)$ where V_f is a set of control nodes, $E_f \subseteq V_f \times \text{ops} \times V_f$ is a set of labeled directed edges labeled by operations from ops (defined below), and entry_f and exit_f are nodes in V_f denoting the entry and exit nodes of a CFG. Each edge label is either an action a , a call $\text{call}(g)$ to a procedure g , an asynchronous call $\text{async}(g)$ to a procedure g , or a finish $\text{finish}(g)$ to a procedure g . A control flow graph representation can be computed from the program syntax using standard compiler techniques [2].

Additionally, we make the simplifying assumption that each label a is used at most once, and that if there is an edge (u, a, v) , then the node u has no other outgoing edges. Thus, the node u uniquely determines that the action a is about to be executed.

As usual, we assume $V_f \cap V_g = \emptyset$ for two distinct procedures f, g . Let $V = \cup\{V_f \mid f \text{ is a procedure}\}$ and $E = \cup\{E_f \mid f \text{ is a procedure}\}$.

We now define a CDPN M_G from a CFG representation \mathcal{G} . The set of actions consists of all actions a in ops , together with a new “silent” action τ . The set of control locations is $\{\#, \hat{\#}, \text{done}\}$. The set of stack symbols $\Gamma = V \cup \{\text{wait}[u, g, v] \mid (u, \text{finish}(g), v) \in E\}$. Intuitively, we will use the stack to maintain the program stack, with the topmost symbol being the current control node. The control locations $\#$ and $\hat{\#}$ are “dummy” locations used to orchestrate program steps, with $\hat{\#}$ indicating that the execution is in the context of a finish operation. The control location done will be used to indicate that a child process has terminated.

Now for the transitions in Δ . For each $(u, a, v) \in E$, we have the rules $P^* : \#u \xrightarrow{a} \#v$ and $P^* : \hat{\#}u \xrightarrow{a} \hat{\#}v$. For each edge $(u, \text{call}(g), v)$, we have the rules $P^* : \#u \xrightarrow{\tau} \# \text{entry}_g v$ and $P^* : \hat{\#}u \xrightarrow{\tau} \hat{\#} \text{entry}_g v$. For each edge $(u, \text{async}(g), v)$, we have the rules $P^* : \#u \xrightarrow{\tau} \#v \triangleright \# \text{entry}_g$ and $P^* : \hat{\#}u \xrightarrow{\tau} \hat{\#}v \triangleright \hat{\#} \text{entry}_g$. To model returns from a procedure g , we add the rules $P^* : \# \text{exit}_g \xrightarrow{\tau} \#$ and $P^* : \hat{\#} \text{exit}_g \xrightarrow{\tau} \hat{\#}$. For each edge $(u, \text{finish}(g), v)$, we add the rule $P^* : \#u \xrightarrow{\text{tau}} \# \text{wait}[u, g, v] \triangleright \hat{\#} \text{entry}_g$.

Next, we give the rules performing the synchronization at the end of a finish statement. The first rule, $\text{done}^* : \#\$ \xrightarrow{\tau} \text{done}$, encodes that in finish mode, a process goes to the control state done when all its children terminate. The second rule, $P^* \text{done} : \# \text{wait}[u, p, v] \xrightarrow{\tau} \#v$, encodes that the “top level” finish call finishes when the entire finish (spawned as its youngest child) finishes. These

two rules ensure that a process makes progress beyond a $\text{finish}(g)$ statement only when all processes spawned transitively from g terminate.

It is easy to see that for every CFG \mathcal{G} , the CDPN $M_{\mathcal{G}}$ preserves all the behaviors of \mathcal{G} . Moreover, $M_{\mathcal{G}}$ is *linear* in the size of \mathcal{G} .

To solve the MHP decision problem, we perform symbolic backward reachability of the CDPN using the algorithm of [4]. Infinite sets of M -configurations are represented using *M-tree automata*. Let $M = (A, P, \Gamma, \Delta)$ be a CDPN. An M -tree automaton (Q, F, δ) consists of a finite set Q of states, a set $F \subseteq Q$ of final states, and a set δ of rules of the following two forms: (a) $\gamma(q) \rightarrow q'$, where $\gamma \in \Gamma$, and $q, q' \in Q$, and (b) $p(L) \rightarrow q$ where $p \in P$, $q \in Q$, and L is a regular language over Q .

We define the relation \rightarrow_{δ} between terms over $P \cup \Gamma \cup Q$ as: $t \rightarrow_{\delta} t'$ iff there exists a context C and a rule $r \in \delta$ such that $t = C[s]$, $t' = C[s']$, and (1) either $r = \gamma(q) \rightarrow q'$ and $s = \gamma(q)$ and $s' = q'$, or (2) $r = p(L) \rightarrow q$, $s = p(q_1, \dots, q_n)$, $q_1 \dots q_n \in L$, and $s' = q$. A term t is accepted by the M -tree automaton if $t \rightarrow_{\delta}^* q$ for some $q \in F$, where \rightarrow_{δ}^* is the reflexive transitive closure of \rightarrow_{δ} . The language of an M -tree automaton is the set of all terms accepted by it.

In particular, for the MHP problem, we are given two actions a and a' and are interested in all M -configurations in which there exist two processes, one about to execute a and the other about to execute a' . By our assumption on CFGs, there are unique nodes u_a and $u_{a'}$ which signify actions a and a' are about to be executed respectively. With abuse of notation, we refer to these nodes as a and a' respectively.

We now give an M -tree automaton $\mathcal{A}_{a,a'} = (Q, F, \delta)$ that recognizes every such configuration. We set $Q = \{q_0, q_a, q_{a'}, q_{ok}\}$. Intuitively, q_0 says we have not seen either a or a' , q_a says we have seen a (but not a') and $q_{a'}$ symmetrically, and q_{ok} says we have seen both. The only final state is q_{ok} . Intuitively, the automaton traverses a term bottom-up, remembering if it has seen an a , an a' or both in its control state. It accepts if at the root it has seen both a and a' . Formally, the rules in δ are the following. First, the rules to go over the stack symbols: $a(q_0) \rightarrow q_a$, $a'(q_0) \rightarrow q_{a'}$, $a(q_{a'}) \rightarrow q_{ok}$, $a'(q_a) \rightarrow q_{ok}$, and $\gamma(q) \rightarrow q$ for every $\gamma \notin \{a, a'\}$ and $q \in Q$. Next, the rules to go over the locations: $p(Q^*q_aQ^*) \rightarrow q_a$, $p(Q^*q_{a'}Q^*) \rightarrow q_{a'}$, $p(Q^*q_aQ^*q_{a'}Q^*) \rightarrow q_{ok}$, $p(Q^*q_{a'}Q^*q_aQ^*) \rightarrow q_{ok}$, $p(Q^*) \rightarrow q_0$ for $p \in \{\#, \hat{\#}\}$, and finally, $\text{done}(Q^*) \rightarrow q_0$.

Given the CDPN $M_{\mathcal{G}}$ for a CFG \mathcal{G} and the M -tree automaton $\mathcal{A}_{a,a'}$, the results of [4] show that $\text{pre}^*(\mathcal{A}_{a,a'})$ can be computed in time linear in the size of $M_{\mathcal{G}}$. (Note that the size of $\mathcal{A}_{a,a'}$ is constant, and there is a fixed constant number of distinct (backward deterministic) regular expressions in $M_{\mathcal{G}}$ and $\mathcal{A}_{a,a'}$.) To check if $\text{MHP}(a, a')$, we check if the initial configuration $\#entry_{main}$ is in $\text{pre}^*(\mathcal{A}_{a,a'})$. The following theorem follows.

Theorem 2. *The single-query MHP decision problem for programs with procedures can be solved in time linear in the size of a program.*

Since the number of pairs of program locations is quadratic in the size of the program, this implies a cubic algorithm for the MHP computation problem.

5 Type System

We now return to studying Featherweight X10 without procedures, exactly as defined in Section 2. We will here present a type system that we in the following section will show gives a characterization of the MHP problem.

We define

$$\begin{aligned} \text{symcross} &: \text{LabelSet} \times \text{LabelSet} \rightarrow \text{LabelPairSet} \\ \text{symcross}(B_1, B_2) &= (B_1 \times B_2) \cup (B_2 \times B_1) \end{aligned}$$

We need *symcross* to help produce a symmetric set of pairs of labels.

We will use judgments of the form $\vdash s : M, O, L$. Here, s is a statement, M is a set of label pairs, and O and L are sets of labels. The meaning of $\vdash s : M, O, L$ is that s has MHP information M , that while s is executing statements with labels in L will be executed, and when s terminates, statements with labels in O may still be executing. Here are the six rules for deriving such judgments.

$$\frac{\vdash s_1 : M_1, O_1, L_1 \quad \vdash s_2 : M_2, O_2, L_2}{\vdash s_1 ; s_2 : M_1 \cup M_2 \cup \text{symcross}(O_1, L_2), O_1 \cup O_2, L_1 \cup L_2} \quad (8)$$

$$\frac{\vdash s : M, O, L}{\vdash \text{loop } s : M \cup \text{symcross}(O, L), O, L} \quad (9)$$

$$\frac{\vdash s : M, O, L}{\vdash \text{async } s : M, L, L} \quad (10)$$

$$\frac{\vdash s : M, O, L}{\vdash \text{finish } s : M, \emptyset, L} \quad (11)$$

$$\vdash a^l : \emptyset, \emptyset, \{l\} \quad (12)$$

$$\vdash \text{skip} : \emptyset, \emptyset, \emptyset \quad (13)$$

The following four theorems are standard and have straightforward proofs.

Theorem 3. (Existence of Typing) *There exists M, O, L such that $\vdash s : M, O, L$.*

Proof. Straightforward induction on s .

Theorem 4. (Unique Typing) *If $\vdash s : M_1, O_1, L_1$ and $\vdash s : M_2, O_2, L_2$, then $M_1 = M_2$ and $O_1 = O_2$ and $L_1 = L_2$.*

Proof. Straightforward induction on s .

Theorem 5. (Subject Reduction) *If $\vdash R : M, O, L$ and $R \rightarrow s'$, then there exists M', O', L' such that $\vdash s' : M', O', L'$ and $M' \subseteq M$ and $O' \subseteq O$ and $L' \subseteq L$.*

Proof. Straightforward case analysis of $R \rightarrow s'$.

Theorem 6. (Preservation) *If $\vdash s : M, O, L$ and $s \mapsto s'$, then there exists M', O', L' such that $\vdash s' : M', O', L'$ and $M' \subseteq M$ and $O' \subseteq O$ and $L' \subseteq L$.*

Proof. From $s \mapsto s'$ we have that there exist a context C and a redex R such that $s = C[R]$, and that there exists s'' such that $C[R] \mapsto C[s'']$ and $R \rightarrow s''$. We proceed by induction on the derivation of C .

$C = C_1 ; s_1$. We have $s = C[R] = (C_1 ; s_1)[R] = (C_1[R]) ; s_1$. The result is immediate from the induction hypothesis.

$C = P ; C_1$. The result is immediate from the induction hypothesis.

$C = \text{async } C_1$. The result is immediate from the induction hypothesis.

$C = \text{finish } C_1$. The result is immediate from the induction hypothesis.

$C = \square$. We have $C[R] = R$ and since $s = C[R]$, we have $s = R$. So from $\vdash s : M, O, L$ we have $\vdash R : M, O, L$. Additionally $s' = C[s''] = s''$. From $\vdash R : M, O, L$ and $R \rightarrow s''$ and Theorem 5, we have the desired result.

6 Correctness

We will now prove that our type system characterizes the MHP analysis problem, for Featherweight X10 without procedures.

6.1 A Characterization of CBE

Let us define $sCBE(s)$ and the helper function $runnable(s)$ as follows:

$$sCBE(s_1 ; s_2) = \begin{cases} sCBE(s_1) \cup sCBE(s_2) \cup \\ \text{syncross}(runnable(s_1), runnable(s_2)) & \text{if } s_1 = P \\ sCBE(s_1) & \text{otherwise} \end{cases} \quad (14)$$

$$sCBE(\text{loop } s) = \emptyset \quad (15)$$

$$sCBE(\text{async } s) = sCBE(s) \quad (16)$$

$$sCBE(\text{finish } s) = sCBE(s) \quad (17)$$

$$sCBE(a^l) = \emptyset \quad (18)$$

$$sCBE(\text{skip}) = \emptyset \quad (19)$$

$$sCBE(s) = \emptyset \quad (20)$$

$$runnable(s_1 ; s_2) = \begin{cases} runnable(s_1) \cup runnable(s_2) & \text{if } s_1 = P \\ runnable(s_1) & \text{otherwise} \end{cases} \quad (21)$$

$$runnable(\text{loop } s) = \emptyset \quad (22)$$

$$runnable(\text{async } s) = runnable(s) \quad (23)$$

$$runnable(\text{finish } s) = runnable(s) \quad (24)$$

$$runnable(a^l) = \{l\} \quad (25)$$

$$runnable(\text{skip}) = \emptyset \quad (26)$$

Lemma 1. *There exists C such that $C[a^l] = s$ if and only if $l \in runnable(s)$.*

Proof. \leftarrow) We must show that if $l \in \text{runnable}(s)$ then there exists C such that $C[a^l] = s$.

Let us perform induction on s and examine the six cases.

If $s \equiv s' ; s''$ we have two cases to consider when $l \in \text{runnable}(s)$ from Rule (21).

Suppose $s' = P'$. From this premise we must consider the cases of $l \in \text{runnable}(P')$ and $l \in \text{runnable}(s'')$.

Suppose $l \in \text{runnable}(P')$. Then from the induction hypothesis we have $C'[a^l] = P'$. We choose $C = C' ; s''$ and since $C'[a^l] = P'$ we have $C[a^l] = s$ as desired.

Suppose $l \in \text{runnable}(s'')$. Then from the induction hypothesis we have $C''[a^l] = s''$. We choose $C = P' ; C''$ and since $C''[a^l] = s''$ have $C[a^l] = s$ as desired.

Suppose $s' \neq P'$. Then from this premise we have 1) $l \in \text{runnable}(s')$. Using the induction hypothesis we have that there exists C' such that 2) $C'[a^l] = s'$. We choose $C = C' ; s''$ and from 2) we have $C[a^l] = s$ as desired.

If $s \equiv \text{loop } s'$ then $\text{runnable}(s) = \emptyset$ which contradicts our premise which makes this case vacuously true.

If $s \equiv \text{async } s'$ then from Rule (23) we have 1) $l \in \text{runnable}(s')$. We then use the induction hypothesis with 1) to get that there exists C' such that 2) $C'[a^l] = s'$. We choose $C = \text{async } C'$ and with 2) we have $C[a^l] = s$ as desired.

If $s \equiv \text{finish } s'$ then we use similar reasoning as the previous case.

If $s \equiv a^l$ then we choose $C = \square$ and thus have $C[a^l] = s$ as desired.

If $s \equiv \text{skip}$ then $\text{runnable}(s) = \emptyset$ which contradicts our premise which makes this case vacuously true.

\rightarrow) We must show that if there exists C such that $C[a^l] = s$ then $l \in \text{runnable}(s)$.

Let us perform induction on s and examine the six cases.

If $s \equiv s' ; s''$ then there are two productions of C that conform to our premise: $C = C' ; s''$ and $C'[a^l] = s'$ or $C = P' ; C''$ and $C''[a^l] = s''$.

Suppose $C = C' ; s''$ and $C'[a^l] = s'$. Then Rules (21) and (21) may apply and we must show $l \in \text{runnable}(s)$ from both rules. From this premise we use the induction hypothesis to get $l \in \text{runnable}(s')$. In either case of $s' = P'$ or $s' \neq P'$, combining this with Rule (21) give us the desired conclusion of $l \in \text{runnable}(s)$.

Suppose $C = P' ; C''$ and $C''[a^l] = s''$. Then the first case of Rule (21) only applies to this case as $s' = P'$. Using the induction hypothesis with this premise gives us $l \in \text{runnable}(s'')$. Using this with Rule (21) gives us $l \in \text{runnable}(s)$ as desired.

If $s \equiv \text{loop } s'$ then we see that there is no C such that $C[a^l] = s$ which contradicts our premise and makes this case vacuously true.

If $s \equiv \text{async } s'$ then we see that our premise is true only if there exists C' such that $C = \text{async } C'$ and $C'[a^l] = s'$. We use our induction hypothesis to get that $l \in \text{runnable}(s')$. Combining this with Rule (23) gives use $l \in \text{runnable}(s)$ as desired.

If $s \equiv \text{finish } s'$ then we use similar reasoning as the previous case.

If $s \equiv a^l$ then from Rule (25) we see our conclusion is true.

If $s \equiv skip$ then we see that there is no C such that $C[a^l] = s$ which contradicts our premise and makes this case vacuously true.

Lemma 2. $CBE(s, l_1, l_2)$ if and only if $(l_1, l_2) \in sCBE(s)$.

Proof. \leftarrow) We must show if $(l_1, l_2) \in sCBE(s)$ then $CBE(s, l_1, l_2)$.

Let us perform induction on s . There are six cases to examine.

If $s \equiv s' ; s''$ then there are two cases from Rule (14) to consider on how $(l_1, l_2) \in sCBE(s)$.

Suppose $s' = P'$ from Rule (14). Then either $(l_1, l_2) \in sCBE(P')$, $(l_1, l_2) \in sCBE(s'')$, or $(l_1, l_2) \in symcross(runnable(P'), runnable(s''))$.

Suppose $(l_1, l_2) \in sCBE(P')$. Then using the induction hypothesis with this premise gives us 1) $CBE(P', l_1, l_2)$. From the definition of CBE we have that there exists C'_1 and C'_2 such that 2) $P' = C'_1[a^{l_1}]$, 3) $P' = C'_2[a^{l_2}]$ and 4) $C'_1 \neq C'_2$. Let 5) $C_1 = C'_1 ; s''$ and 6) $C_2 = C'_2 ; s''$. From 4),5) and 6) we have 7) $C_1 \neq C_2$. We combine 2) with 5) and 3) with 6) to get 8) $s = C_1[a^{l_1}]$ and 9) $s = C_2[a^{l_2}]$. From the definition of CBE and 7),8) and 9) we have $CBE(s, l_1, l_2)$ as desired.

Suppose $(l_1, l_2) \in sCBE(s'')$. Then using the induction hypothesis with this premise gives us 1) $CBE(s'', l_1, l_2)$. From the definition of CBE we have that there exists C''_1 and C''_2 such that 2) $s'' = C''_1[a^{l_1}]$, 3) $s'' = C''_2[a^{l_2}]$ and 4) $C''_1 \neq C''_2$. Let 5) $C_1 = P' ; C''_1$ and 6) $C_2 = P' ; C''_2$. From 4),5) and 6) we have 7) $C_1 \neq C_2$. We combine 2) with 5) and 3) with 6) to get 8) $s = C_1[a^{l_1}]$ and 9) $s = C_2[a^{l_2}]$. From the definition of CBE and 7),8) and 9) we have $CBE(s, l_1, l_2)$ as desired.

Suppose $(l_1, l_2) \in symcross(runnable(P'), runnable(s''))$. Then from the definition of $symcross()$ we have either 1) $l_1 \in runnable(P')$ and 2) $l_2 \in runnable(s'')$ or vice versa. In either case we proceed using similar reasoning. Using Lemma (13) with 1) and 2) gives us that there exists C'_1 and C'_2 such that 3) $C'_1[a^{l_1}] = P'$ and 4) $C'_2[a^{l_2}] = s''$. Let 5) $C_1 = C'_1 ; s''$ and 6) $C_2 = P' ; C'_2$. From 5) and 6) we immediately see 7) $C_1 \neq C_2$. Combining 3) with 5) and 4) with 6) gives us 8) $C_1[a^{l_1}] = s$ and 9) $C_2[a^{l_2}] = s$. From the definition of CBE with 7),8) and 9) we have $CBE(s, l_1, l_2)$ as desired.

Suppose $s' \neq P'$. Then we use similar reasoning as the case where $(l_1, l_2) \in sCBE(P')$.

If $s \equiv loop\ s'$ then from Rule (15) we have $sCBE(s) = \emptyset$ which contradicts our premise making this case vacuously true.

If $s \equiv async\ s'$ then from Rule (16) we have 1) $CBE(s', l_1, l_2)$. From the definition of CBE we have that there exists C'_1 ; and C'_2 such that 2) $s' = C'_1[a^{l_1}]$, 3) $s' = C'_2[a^{l_2}]$ and 4) $C_1 \neq C_2$. Let 5) $C_1 = async\ C'_1$ and 6) $C_2 = async\ C'_2$. From 4),5) and 6) we have 7) $C_1 \neq C_2$. We combine 2) with 5) and 3) with 6) to get 8) $s = C_1[a^{l_1}]$ and 9) $s = C_2[a^{l_2}]$. From the definition of CBE and 7),8) and 9) we have $CBE(s, l_1, l_2)$ as desired.

If $s \equiv finish\ s'$ then we proceed using similar reasoning as the previous case.

If $s \equiv a^l$ then from Rule (18) we have $sCBE(s) = \emptyset$ which contradicts our premise making this case vacuously true.

If $s \equiv skip$ then we use similar reasoning as the previous case.

→) We must show if $\text{CBE}(s, l_1, l_2)$ then $(l_1, l_2) \in \text{sCBE}(s)$.

Let us perform induction on s . There are six cases to examine.

If $s \equiv s' ; s''$ then from the definition of CBE we have that there exists C_1 and C_2 such that a) $s = C_1[a^{l_1}]$, b) $s = C_2[a^{l_2}]$ and c) $C_1 \neq C_2$. Let us perform case analysis on C_1 and C_2 observing that the $C ; s$ and $P ; C$ productions are the only ones that may satisfy a) and b).

Suppose $C_1 = C'_1 ; s''$ and $C_2 = C'_2 ; s''$. Since Rule (14) has two cases, we must consider if $s' = P'$ or $s' \neq P'$. Combining this premise with c) gives us 1) $C'_1 \neq C'_2$. Additionally from this premise and a) and b) we obtain 2) $s' = C'_1[a^{l_1}]$ and 3) $s' = C'_2[a^{l_2}]$. Using the definition of CBE with 1), 2) and 3) we have 4) $\text{CBE}(s', l_1, l_2)$. We use the induction hypothesis with 4) to get 5) $(l_1, l_2) \in \text{sCBE}(s')$. From the Rule (14) with 5) in either case of $s' = P'$ or $s' \neq P'$ we have $(l_1, l_2) \in \text{sCBE}(s' ; s'')$.

Suppose $C_1 = P' ; C''_1$ and $C_2 = P' ; C''_2$. Then combining this premise with c) gives us 1) $C''_1 \neq C''_2$. Additionally from our this premise and a) and b) we obtain 2) $s'' = C''_1[a^{l_1}]$ and 3) $s'' = C''_2[a^{l_2}]$. Using the definition of CBE with 1), 2) and 3) we have 4) $\text{CBE}(s'', l_1, l_2)$. We use the induction hypothesis with 4) to get 5) $(l_1, l_2) \in \text{sCBE}(s'')$. From the Rule (14) combined with 5) we have $(l_1, l_2) \in \text{sCBE}(s)$ as desired.

Suppose $C_1 = C'_1 ; s''$ and $C_2 = P' ; C''_2$. Combining this premise with a) and b) gives us 1) $P' = C'_1[a^{l_1}]$ and 2) $s'' = C''_2[a^{l_2}]$. From applying Lemma (1) with 1) and 2) we obtain 3) $l_1 \in \text{runnable}(P')$ and 4) $l_2 \in \text{runnable}(s'')$. From the definition of *syncross*() we have 5) $(l_1, l_2) \in \text{syncross}(\text{runnable}(P'), \text{runnable}(s''))$. Using Rule (14) with 5) gives us $(l_1, l_2) \in \text{sCBE}(s)$ as desired.

Suppose $C_1 = P ; C''_1$ and $C_2 = C'_2 ; s''$. We proceed using similar reasoning as the previous case.

If $s \equiv \text{loop } s'$ we see that there is no C such that $C[a^{l_1}] = s$ and thus from its definition, we have $\text{CBE}(s, l_1, l_2) = \text{false}$ which contradicts our premise and making this case vacuously true.

If $s \equiv \text{async } s'$ then from the definition of CBE we have that there exists C_1 and C_2 such that 1) $s = C_1[a^{l_1}]$, 2) $s = C_2[a^{l_2}]$ and 3) $C_1 \neq C_2$. We observe that there is only context production that we may use with 1) and 2) to get 4) $s = (\text{async } C'_1)[a^{l_1}]$ and 5) $s = (\text{async } C'_2)[a^{l_2}]$. We see from 3), 4) and 5) that 6) $C'_1 \neq C'_2$. From 4) and 5) we may obtain 7) $s' = C'_1[a^{l_1}]$ and 8) $s' = C'_2[a^{l_2}]$. From the definition of CBE and 6), 7) and 8) we have 9) $\text{CBE}(s', l_1, l_2)$. Using the induction hypothesis and 9) we have 10) $(l_1, l_2) \in \text{sCBE}(s')$. Using Rule (16) we have $(l_1, l_2) \in \text{sCBE}(s)$ as desired.

If $s \equiv \text{finish } s'$ then we use similar reasoning as the previous case.

If $s \equiv a^l$ then we see that there is exactly one C such that $C[a^l] = s$ where $C = \square$. From its definition then we see that $\text{CBE}(s) = \text{false}$ and contradicts our premise making this case vacuously true.

If $s \equiv \text{skip}$ then we use similar reasoning as the loop case.

Theorem 7. (CBE Characterization) $\text{CBE}(s) = \text{sCBE}(s)$.

Proof. Follows from Lemma (2) and the definition of CBE().

6.2 Equivalence of MHP and Types

We will give a type-based characterization of the May Happen in Parallel problem. For a statement s we have from Theorem 3 and Theorem 4 that we have that there exist unique M, O, L such that $\vdash s : M, O, L$. We define

$$\text{MHP}_{type}(s) = M \quad (\text{where } \vdash s : M, O, L)$$

We use the subscript *type* to emphasize that the definition is type based. We will show that for all s , $\text{MHP}_{sem}(s) = \text{MHP}_{type}(s)$ (Theorem 10).

Lemma 3. *If $\vdash s : M, O, L$ and $\text{runnable}(s) \subseteq L$.*

Proof. Let us perform induction on s and examine the six cases.

If $s \equiv s_1 ; s_2$ then from the definition of $\text{runnable}()$ we have two cases to consider from Rule (21): either $\text{runnable}(s) = \text{runnable}(s_1) \cup \text{runnable}(s_2)$ and $s = P$ or $\text{runnable}(s) = \text{runnable}(s_1)$.

Suppose $\text{runnable}(s) = \text{runnable}(s_1) \cup \text{runnable}(s_2)$ and $s = P$. From Rule (8) 1) $\vdash s_1 : M_1, O_1, L_1$, 2) $\vdash s_2 : M_2, O_2, L_2$ and 3) $L = L_1 \cup L_2$. Using the induction hypothesis on 1) and 2) gives us 4) $\text{runnable}(s_1) \subseteq L_1$ and 5) $\text{runnable}(s_2) \subseteq L_2$. Combining this premise with 3), 4) and 5) gives us $\text{runnable}(s) \subseteq L$ as desired.

Suppose $\text{runnable}(s) = \text{runnable}(s_1)$. From Rule (8) we have 1) $\vdash s_1 : M_1, O_1, L_1$, 2) $\vdash s_2 : M_2, O_2, L_2$ and 3) $L = L_1 \cup L_2$. Using the induction hypothesis with this premise and 2) gives us 4) $\text{runnable}(s_1) \subseteq L_2$. Combining this premise with 3) and 4) gives us $\text{runnable}(s) \subseteq L$ as desired.

If $s \equiv \text{loop } s_1$ then from the definition of $\text{runnable}()$ we have $\text{runnable}(s) = \emptyset$ which we allows us to easily see that $\text{runnable}(s) \subseteq L$.

If $s \equiv \text{async } s_1$ then from the definition of $\text{runnable}()$ we have 1) $\text{runnable}(s) = \text{runnable}(s_1)$. Substituting 1) in to the premise gives us 2) $l_0 \in \text{runnable}(s_1)$. From Rule (10) we have 3) $\vdash s_1 : M_1, O_1, L_1$ and 4) $L = L_1$. Using the induction hypothesis with 2) and 3) we obtain 5) $l_0 \in L_1$. We substitute 4) in 5) to get $l \in L$ as desired.

If $s \equiv \text{finish } s_1$ then we proceed using similar reasoning as the previous case.

If $s \equiv a^l$ then from the definition of $\text{runnable}()$ we have 1) $\text{runnable}(s) = \{l\}$. From Rule (12) we have 2) $L = \{l\}$. Substituting 2) in 1) gives us 3) $\text{runnable}(s) = L$. From 3) we see that $\text{runnable}(s) \subseteq L$ is true.

If $s \equiv \text{skip}$ then from the definition of $\text{runnable}()$ we have $\text{runnable}(s) = \emptyset$ which we allows us to easily see that $\text{runnable}(s) \subseteq L$.

Lemma 4. *If $\vdash P : M, O, L$ then $\text{runnable}(P) \subseteq O$.*

Proof. Let us perform induction on P there are two cases to consider.

If $P = P_1 ; P_2$ then from the definition of $\text{runnable}()$ we have 1) $\text{runnable}(s) = \text{runnable}(P_1) \cup \text{runnable}(P_2)$. From Rule (8) we have 2) $\vdash P_1 : M_1, O_1, L_1$, 3) $\vdash P_2 : M_2, O_2, L_2$ and 4) $O = O_1 \cup O_2$. Using the induction hypothesis with

2) and 3) premise gives us 5) $runnable(P_1) \subseteq O_1$ and 6) $runnable(P_2) \subseteq O_2$. Combining 1),4),5) and 6) gives us $runnable(P) \subseteq O$ as desired.

If $P = async\ s_1$ then from Rule (10) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $O = L_1$. From the definition of $runnable()$ we have 3) $runnable(P) = runnable(s_1)$. Using Lemma (3) with 1) and 4) we have 4) $runnable(s_1) \subseteq L_1$. Substituting 2) and 3) in 4) gives us $runnable(P) \subseteq O$ as desired.

Theorem 8. *If $\vdash s : M, O, L$, then $CBE(s) \subseteq M$.*

Proof. Let us perform induction on s and examine the six cases.

If $s \equiv s_1 ; s_2$ then from Theorem (1) and Rule (14) we have either $CBE(s) = CBE(s_1) \cup CBE(s_2) \cup syncross(runnable(s_1), runnable(s_2))$ where $s_1 = P$ or $CBE(s) = CBE(s_1)$.

Suppose $CBE(s) = CBE(s_1) \cup CBE(s_2) \cup syncross(runnable(s_1), runnable(s_2))$ where $s_1 = P$. From Rule (8) we have 1) $\vdash s_1 : M_1, O_1, L_1$, 2) $\vdash s_2 : M_2, O_2, L_2$ and 3) $M = M_1 \cup M_2 \cup syncross(O_1, L_2)$. Using the induction hypothesis with 1) and 2) we obtain 4) $CBE(s_1) \subseteq M_1$ and 5) $CBE(s_2) \subseteq M_2$. Using Lemma (3) this premise and 1) gives us 6) $runnable(s_1) \subseteq O_1$. We use Lemma (4) with 2) to get 7) $runnable(s_2) \subseteq L_2$. From 6) and 7) and the definition of $syncross()$ we have that 8) $syncross(runnable(s_1), runnable(s_2)) \subseteq syncross(O_1, L_2)$. Combining 3),4),5) and 8) we obtain $CBE(s) \subseteq M$ as desired.

Suppose $CBE(s) = CBE(s_1)$. From Rule (8) we have 1) $\vdash s_1 : M_1, O_1, L_1$, 2) $\vdash s_2 : M_2, O_2, L_2$ and 3) $M = M_1 \cup M_2 \cup syncross(O_1, L_2)$. Using the induction hypothesis with 1) we get 4) $CBE(s_1) \subseteq M_1$. Substituting this premise with 4) gives us 5) $CBE(s) \subseteq M_1$. Combining 3) and 5) gives us $CBE(s) \subseteq M$ as desired.

If $s \equiv loop\ s_1$ then from Theorem (1) and Rule (15) we have $CBE(s) = \emptyset$. From this we immediatly may see that our conclusion is true.

If $s \equiv async\ s_1$ then from Theorem (1) and Rule (16) we have 1) $CBE(s) = CBE(s_1)$. From Rule (10) we have 2) $\vdash s_1 : M_1, O_1, L_1$ and 3) $M = M_1$. Using the induction hypothesis with 2) we obtain 4) $CBE(s_1) \subseteq M_1$. Substituting 1) and 3) in 4) gives us $CBE(s) \subseteq M$ as desired.

If $s \equiv finish\ s_1$ then we proceed using reasoning similar to the previous case.

If $s \equiv a^l$ then we proceed using similar reasoning as the loop case.

If $s \equiv skip$ then we proceed using similar reasoning as the loop case.

The following theorem can be proved with the technique used by Lee and Palsberg [9] to prove a similar result.

Theorem 9. (Overapproximation) $MHP_{sem}(s) \subseteq MHP_{type}(s)$.

Proof. From the definition of $MHP_{type}(s)$ we have $MHP_{type}(s) = M$, where $\vdash s : M, O, L$. From the definition of $MHP_{sem}(s)$ we have $MHP_{sem}(s) = \bigcup_{s' : s \mapsto^* s'} CBE(s')$. Suppose $s \mapsto^* s'$. It is sufficient to show $CBE(s') \subseteq M$. From Theorem 6 and induction on the number of steps in $s \mapsto^* s'$, we have M', O', L' such that $\vdash s' : M', O', L'$ and $M' \subseteq M$ and $O' \subseteq O$ and $L' \subseteq L$. From Theorem 8 we have $CBE(s') \subseteq M'$. From $CBE(s') \subseteq M'$ and $M' \subseteq M$, we conclude $CBE(s') \subseteq M$.

We now embark on proving the dual of Theorem 9. We begin with three lemmas that can be proved easily by induction.

Lemma 5. *If $s_1 \mapsto^* s'_1$ then $s_1 ; s_2 \mapsto^* s'_1 ; s_2$.*

Proof. It is sufficient to prove this by showing if $s_1 \mapsto^n s'_1$ then $s_1 ; s_2 \mapsto^n s'_1 ; s_2$.

We will now show that if $s_1 \mapsto^n s'_1$ then $s_1 ; s_2 \mapsto^n s'_1 ; s_2$.

We perform induction on n . If $n = 0$ then $s'_1 = s_1$ and $s_1 ; s_2 \mapsto^0 s_1 ; s_2$ is immediately obvious.

If $n = i + 1$ and we have 1) $s_1 \mapsto s''_1$ and 2) $s''_1 \mapsto^i s'_1$. From 1) we have that there exists a context C_1 and redex R such that 3) $s_1 = C_1[R_1]$, 4) $s''_1 = C_1[s''_1]$ and 5) $R_1 \rightarrow s''_1$. Let 6) $C = C_1 ; s_2$. Then from the definition of \mapsto and 5) we have 7) $C[R_1] \mapsto C[s''_1]$. Using 3) and 4) with 6) and 7) gives us 8) $s_1 ; s_2 \mapsto s''_1 ; s_2$. Using the induction hypothesis with 2) we have 9) $s''_1 ; s_2 \mapsto^i s'_1 ; s_2$ which we combine with 8) to get $s_1 ; s_2 \mapsto^{i+1} s'_1 ; s_2$ as desired.

Lemma 6. *If $s \mapsto^* s'$ then $async\ s \mapsto^* async\ s'$.*

Proof. We use similar reasoning as Lemma (5).

Lemma 7. *If $s \mapsto^* s'$ then $finish\ s \mapsto^* finish\ s'$.*

Proof. We use similar reasoning as Lemma (5).

Lemma 8. *For all s , $s \mapsto^* skip$.*

Proof. We perform induction on s and examine the six cases.

If $s \equiv s_1 ; s_2$ then using the induction hypothesis we have 1) $s_1 \mapsto^* skip$ and 2) $s_2 \mapsto^* skip$. Using Lemma (5) with 1) gives us 3) $s_1 ; s_2 \mapsto^* skip ; s_2$. We use $C = \square$ and $R = skip ; s_2$ with Rule (1) to get 4) $skip ; s_2 \mapsto skip$. We combine 3) and 4) to get $s_1 ; s_2 \mapsto^* skip$ as desired.

If $s \equiv loop\ s_1$ then using $C = \square$ and $R = loop\ s_1$ with Rule (3) we arrive at our conclusion.

If $s \equiv async\ s_1$ then using the induction hypothesis we have 1) $s_1 \mapsto^* skip$. Using Lemma (6) with 1) yields 2) $async\ s_1 \mapsto^* async\ skip$. We use $C = \square$ and $R = async\ skip$ with Rule (5) to get 3) $async\ skip \mapsto skip$. We combine 2) and 3) to obtain $async\ s_1 \mapsto^* skip$ as desired.

If $s \equiv finish\ s_1$ then we proceed using similar reasoning as the previous case.

If $s \equiv a^l$ then we use $C = \square$ and $R = a^l$ with Rule (7) to obtain our conclusion.

If $s \equiv skip$ the conclusion is immediate.

Lemma 9. *If $s_2 \mapsto^* s'_2$ then $s_1 ; s_2 \mapsto^* s_1 ; s'_2$.*

Proof. From Lemma (8) we have 1) $s_1 \mapsto^* skip$. Using Lemma (5) with 1) gives us 2) $s_1 ; s_2 \mapsto^* skip ; s_2$. Using $C = \square$ and $R = skip ; s_2$ with Rule (1) we have 3) $skip ; s_2 \mapsto s_2$. Combining our premise with 2) and 3) gives us $s_1 ; s_2 \mapsto^* s'_2$ as desired.

Lemma 10. *If $s \mapsto^* s'$ then $P ; s \mapsto^* P ; s'$.*

Proof. It is sufficient to prove this by showing if $s \mapsto^n s'$ then $P ; s \mapsto^n P ; s'$.

We will now show that if $s \mapsto^n s'$ then $P ; s \mapsto^n P ; s$.

We perform induction on n . If $n = 0$ then $s = s'$ and $P ; s \mapsto^0 P ; s$ is immediately obvious.

If $n = i + 1$ and we have 1) $s \mapsto s''$ and 2) $s'' \mapsto^i s'$. From 1) we have that there exists a context C_1 and redex R such that 3) $s = C[R]$, 4) $s'' = C[s''']$ and 5) $R \rightarrow s'''$. Let 6) $C' = P ; C$. Then from the definition of \mapsto and 5) we have 7) $C'[R] \mapsto C'[s''']$. Using 3) and 4) with 6) and 7) gives us 8) $P ; s \mapsto P ; s''$. Using the induction hypothesis with 2) we have 9) $P ; s'' \mapsto^i P ; s'$ which we combine with 8) to get $P ; s \mapsto^{i+1} P ; s'$ as desired.

Lemma 11. *If $s_1 \mapsto^* P_1$ and $s_2 \mapsto^* s'_2$ then $s_1 ; s_2 \mapsto^* P_1 ; s'_2$.*

Proof. Using Lemma (5) with our premise gives us 1) $s_1 ; s_2 \mapsto^* P_1 ; s_2$. Using the premise with Lemma (10) yields 2) $P_1 ; s_2 \mapsto^* P_1 ; s'_2$. Combining 1) and 2) results in $s_1 ; s_2 \mapsto^* P_1 ; s'_2$ as desired.

Lemma 12. *If $s_1 \mapsto^* P_1$ then $s_1 ; s_2 \mapsto^* P_1$.*

Proof. From Lemma (8) we have 1) $s_2 \mapsto^* skip$. Using the premise and 1) with Lemma (11) gives us 2) $s_1 ; s_2 \mapsto^* P_1 ; skip$. Using $C = \square$ and $R = P_1 ; skip$ with Rule (2) gives us 3) $P_1 ; skip \mapsto P_1$. Combining 2) and 3) results in $s_1 ; s_2 \mapsto^* P_1$ as desired.

Lemma 13. *If $\vdash s : M, O, L$ and $l \in L$ then there exists s' such that $s \mapsto^* s'$ and $l \in runnable(s')$.*

Proof. Let us perform induction on s and examine the six cases.

If $s \equiv s_1 ; s_2$ then from Rule (8) we have a) $\vdash s_1 : M_1, O_1, L_1$, b) $\vdash s_2 : M_2, O_2, L_2$ and c) $L = L_1 \cup L_2$. We must consider the cases where $l \in L_1$ or $l \in L_2$.

Suppose $l \in L_1$. Then we use the induction hypothesis with a) and this premise to get that there exists s'_1 such that 1) $s_1 \mapsto^* s'_1$ and 2) $l \in runnable(s'_1)$. Applying Lemma (5) with 1) gives us 3) $s_1 ; s_2 \mapsto^* s'_1 ; s_2$. Using either Rule (21) or (21) both give 4) $l \in runnable(s'_1 ; s_2)$. We choose $s' = s'_1 ; s_2$ and from 3) and 4) we have our conclusion.

Suppose $l \in L_2$. Then we use the induction hypothesis with b) and this premise to get that there exists s'_2 such that 1) $s_2 \mapsto^* s'_2$ and 2) $l \in runnable(s'_2)$. Using Lemma (9) with 1) gives us 3) $s_1 ; s_2 \mapsto^* s'_2$. We choose $s' = s'_2$ and from 2) and 3) we have our conclusion.

If $s \equiv loop\ s_1$ then from Rule (9) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $L = L_1$. Combining 2) with the premise gives us 3) $l \in L_1$. We use the induction

hypothesis with 1) and 3) to get that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $l \in \text{runnable}(s'_1)$. From Lemma (5) with 4) gives us 6) $s_1 ; \text{loop } s_1 \mapsto^* s'_1 ; \text{loop } s_1$. Using $C = \square$ and $R = \text{loop } s_1$ with Rule (4) we have 7) $\text{loop } s_1 \mapsto^* s_1 ; \text{loop } s_1$. Combining 6) with 7) results in 8) $\text{loop } s_1 \mapsto^* s'_1 ; \text{loop } s_1$. Using Rule (21) or (21) with 5) gives us 9) $l \in \text{runnable}(s'_1 ; \text{loop } s_1)$. We choose $s' = s'_1 ; \text{loop } s_1$ and from 8) and 9) we reach our conclusion.

If $s \equiv \text{async } s_1$ then from Rule (10) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $L = L_1$. Combining 2) with the premise gives us 3) $l \in L_1$. We use the induction hypothesis with 1) and 3) to get that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $l \in \text{runnable}(s'_1)$. From Lemma (6) and 4) we obtain 6) $\text{async } s_1 \mapsto^* \text{async } s'_1$. Using Rule (23) with 5) gives us 7) $l \in \text{runnable}(\text{async } s'_1)$. We choose $s' = \text{async } s'_1$ and with 6) and 7) we have our conclusion.

If $s \equiv \text{finish } s_1$ then from Rule (11) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $L = L_1$. Combining 2) with the premise gives us 3) $l \in L_1$. We use the induction hypothesis with 1) and 3) to get that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $l \in \text{runnable}(s'_1)$. From Lemma (7) and 4) we obtain 6) $\text{finish } s_1 \mapsto^* \text{finish } s'_1$. Using Rule (24) with 5) gives us 7) $l \in \text{runnable}(\text{finish } s'_1)$. We choose $s' = \text{finish } s'_1$ and with 6) and 7) we have our conclusion.

If $s \equiv a'$ then from Rule (12) we have $L = \{l'\}$. We combine this with our premise and we have that $l = l'$. From the Rule (25) we have $\text{runnable}(s) = \{l'\}$ and since $l = l'$ we have $l \in \text{runnable}(s)$. We choose $s' = s$ and our conclusion easily follows.

If $s \equiv \text{skip}$ then from Rule (13) we have $L = \emptyset$ which contradicts our premise and makes this case vacuously true.

Lemma 14. *If $\vdash s : M, O, L$ and $l \in O$ then there exists P such that $s \mapsto^* P$ and $l \in \text{runnable}(P)$*

Proof. Let us perform induction on s . There are six cases.

If $s \equiv s_1 ; s_2$ then from Rule (8) we have a) $\vdash s_1 : M_1, O_1, L_1$, b) $\vdash s_2 : M_2, O_2, L_2$ and c) $O = O_1 \cup O_2$. We must consider the case when $l \in O_1$ or when $l \in O_2$.

Suppose $l \in O_1$. Then we use our induction hypothesis with this premise and a) to get that there exists P_1 such that 1) $s_1 \mapsto^* P_1$ and 2) $l \in \text{runnable}(P_1)$. Applying Lemma (12) with 1) gives us 3) $s_1 ; s_2 \mapsto^* P_1$. We choose $P = P_1$ and with 2) and 3) we have our conclusion.

Suppose $l \in O_2$. Then we use the induction hypothesis with this premise and b) to get that there exists P_2 such that 1) $s_2 \mapsto^* P_2$ and 2) $l \in \text{runnable}(P_2)$. We use Lemma (9) with 1) to get 3) $s_1 ; s_2 \mapsto^* P_2$. We choose $P = P_2$ and from 2) and 3) we have our conclusion.

If $s \equiv \text{loop } s_1$ then from Rule (9) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $O = O_1$. Combining 2) with our premise results in 3) $l \in O_1$. We use the induction hypothesis with 1) and 3) to get that there exists P_1 such that 4) $s_1 \mapsto^* P_1$ and 5) $l \in \text{runnable}(P_1)$. Using Lemma (12) with 4) gives us 6) $s_1 ; \text{loop } s_1 \mapsto^* P_1$. We choose $P = P_1$ and with 5) and 6) we have our conclusion.

If $s \equiv \text{async } s_1$ then from Rule (10) we have 1) $\vdash s_1 : M_1, O_1, L_1$ and 2) $O = L_1$. Combining 2) with our premise gives us 3) $l \in L_1$. Using Lemma (13)

with 1) and 3) yields that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $l \in \text{runnable}(s'_1)$. We use Lemma (6) with 4) to get 6) $\text{async } s_1 \mapsto^* \text{async } s'_1$. Combining Rule (23) with 5) gives us 7) $l \in \text{runnable}(\text{async } s'_1)$. We choose $P = \text{async } s'_1$ and from 6) and 7) we have our conclusion.

If $s \equiv \text{finish } s_1$ then from Rule (11) we have $O = \emptyset$ which contradicts our premise. This case is vacuously true.

If $s \equiv a'$ then we use similar reasoning as the previous case.

If $s \equiv \text{skip}$ then we use similar reasoning as the previous case.

Lemma 15. *If $\vdash s : M, O, L$ and $(l_1, l_2) \in M$ then there exists s' such that $s \mapsto^* s'$ and $(l_1, l_2) \in \text{sCBE}(s')$.*

Proof. Let us perform induction on s . This gives us six cases to examine.

If $s \equiv s_1 ; s_2$ then from Rule (8) we have a) $\vdash s_1 : M_1, O_1, L_1$, b) $\vdash s_2 : M_2, O_2, L_2$ and c) $M = M_1 \cup M_2 \cup \text{symcross}(O_1, L_2)$.

Suppose $(l_1, l_2) \in M_1$. Then we may use the induction hypothesis with a) and this premise to get that there exists s'_1 such that 1) $s_1 \mapsto^* s'_1$ and 2) $(l_1, l_2) \in \text{sCBE}(s'_1)$. Using 2) with Rules (14) and (14) both give us 3) $(l_1, l_2) \in \text{sCBE}(s'_1 ; s_2)$. Using Lemma (5) with 1) we have 4) $s_1 ; s_2 \mapsto^* s'_1 ; s_2$. We choose $s' = s'_1 ; s_2$ and 3) and 4) we have our conclusion.

Suppose $(l_1, l_2) \in M_2$. Then we may use the induction hypothesis with b) and this premise to get that there exists s'_2 such that 1) $s_2 \mapsto^* s'_2$ and 2) $(l_1, l_2) \in \text{sCBE}(s'_2)$. Using Lemma (9) with 1) gives us 3) $s_1 ; s_2 \mapsto^* s'_2$. We choose $s' = s'_2$ and from 2) and 3) we have our conclusion.

Suppose $(l_1, l_2) \in \text{symcross}(O_1, L_2)$. Then from the definition of $\text{symcross}()$ and our premise we have either 1) $l_1 \in O_1$ and 2) $l_2 \in L_2$ or vice versa. In either case, we proceed using similar reasoning. We will show for the listed cases. Using Lemma (14) with 1) and a) gives us that there exists P_1 such that 3) $s_1 \mapsto^* P_1$ and 4) $l_1 \in \text{runnable}(P)$. Using Lemma (13) with 2) and b) gives us that there exists s'_2 such that 5) $s_2 \mapsto^* s'_2$ and 6) $l_2 \in \text{runnable}(s'_2)$. From the definition of $\text{symcross}()$ with 4) and 6) we have 7) $(l_1, l_2) \in \text{symcross}(\text{runnable}(P_1), \text{runnable}(s'_2))$. From Rule (14) we have 8) $(l_1, l_2) \in \text{sCBE}(P_1 ; s'_2)$. We use Lemma (11) with 3) and 5) gives us 9) $s_1 ; s_2 \mapsto^* P_1 ; s'_2$. We choose $s' = P_1 ; s'_2$ and from 8) and 9) we obtain our conclusion.

If $s \equiv \text{loop } s_1$ then from Rule (9) we have a) $\vdash s_1 : M_1, O_1, L_1$, b) $M = M_1 \cup \text{symcross}(O_1, L_1)$ and c) $L = L_1$.

Suppose $(l_1, l_2) \in M_1$. Then using the induction hypothesis with a) and this premise gives us that there exists s'_1 such that 1) $s_1 \mapsto^* s'_1$ and 2) $(l_1, l_2) \in \text{sCBE}(s'_1)$. Using 2) with Rules (14) and (14) both gives us 3) $(l_1, l_2) \in \text{sCBE}(s'_1 ; \text{loop } s_1)$. We use Lemma (5) with 1) to get 4) $s_1 ; \text{loop } s_1 \mapsto^* s'_1 ; \text{loop } s_1$. Using $C = \square$ and $R = \text{loop } s_1$ with Rule (4) gives us 5) $\text{loop } s_1 \mapsto^* s_1 ; \text{loop } s_1$. Combining 4) and 5) gives us 6) $\text{loop } s_1 \mapsto^* s'_1 ; \text{loop } s_1$. We choose $s' = s'_1 ; \text{loop } s_1$ and from 3) and 6) we obtain our conclusion.

Suppose $(l_1, l_2) \in \text{symcross}(O_1, L_1)$. Then from the definition of $\text{symcross}()$ and our premise we have either 1) $l_1 \in O_1$ and 2) $l_2 \in L_1$ or vice versa. In either case, we proceed using similar reasoning. Substituting c) in 2) gives us 3)

$l_2 \in L$. Using Lemma (14) with a) and 1) gives us that there exists P_1 such that 4) $s_1 \mapsto^* P_1$ and 5) $l_1 \in \text{runnable}(P_1)$. Using Lemma (13) with our original premise of $\vdash s : M, O, L$ gives us that there exists s'' such that 6) $\text{loop } s \mapsto^* s''$ and 7) $l_2 \in \text{runnable}(s'')$. From the definition of $\text{symcross}()$ with 5) and 7) we have 8) $(l_1, l_2) \in \text{symcross}(\text{runnable}(P_1), \text{runnable}(s''))$. Using Rule (14) with 8) gives us 9) $(l_1, l_2) \in \text{sCBE}(P_1 ; s'')$. Using Lemma (11) with 4) and 6) gives us 10) $s_1 ; \text{loop } s_1 \mapsto^* P_1 ; s''$. Using $C = \square$ and $R = \text{loop } s_1$ with Rule (4) gives us 11) $\text{loop } s_1 \mapsto s_1 ; \text{loop } s_1$. We combine 10) and 11) to get 12) $\text{loop } s_1 \mapsto^* P_1 ; s''$. We choose $s' = P_1 ; s''$ and from 9) and 12) we have our conclusion.

If $s \equiv \text{async } s_1$ then from Rule (10) we have 1) $\vdash s_1 : M_1, O_1, L_1, 2) M = M_1$. We combine the premise with 2) to get 3) $(l_1, l_2) \in M_1$. Using the induction hypothesis with 3) we get that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $(l_1, l_2) \in \text{sCBE}(s'_1)$. From Rule (16) and 5) we get 6) $(l_1, l_2) \in \text{sCBE}(\text{async } s'_1)$. We use Lemma (6) with 4) to get 7) $\text{async } s_1 \mapsto^* \text{async } s'_1$. We choose $s' = \text{async } s'_1$ and from 6) and 7) we have our conclusion.

If $s \equiv \text{finish } s_1$ then from Rule (11) we have 1) $\vdash s_1 : M_1, O_1, L_1, 2) M = M_1$. We combine the premise with 2) to get 3) $(l_1, l_2) \in M_1$. Using the induction hypothesis with 3) we get that there exists s'_1 such that 4) $s_1 \mapsto^* s'_1$ and 5) $(l_1, l_2) \in \text{sCBE}(s'_1)$. From Rule (17) and 5) we get 6) $(l_1, l_2) \in \text{sCBE}(\text{finish } s'_1)$. We use Lemma (7) with 4) to get 7) $\text{finish } s_1 \mapsto^* \text{finish } s'_1$. We choose $s' = \text{finish } s'_1$ and from 6) and 7) we have our conclusion.

If $s \equiv a^l$ then from Rule (12) we have $M = \emptyset$. This contradicts our premise and makes this case vacuously true.

If $s \equiv \text{skip}$ then we use similar reasoning as the previous case.

Lemma 16. (Underapproximation) $\text{MHP}_{\text{type}}(s) \subseteq \text{MHP}_{\text{sem}}(s)$.

Proof. From Lemma (3) we have that there exists M, O and L such that $\vdash s : M, O, L$. Using this with Lemma (15) gives us $M \subseteq \bigcup_{s' : s \mapsto^* s'} \text{sCBE}(s')$. From Theorem (7) we have $M \subseteq \bigcup_{s' : s \mapsto^* s'} \text{CBE}(s')$. From the definition of $\text{MHP}_{\text{type}}(s)$ and $\text{MHP}_{\text{sem}}(s) = \bigcup_{s' : s \mapsto^* s'} \text{CBE}(s')$ we have $\text{MHP}_{\text{type}}(s) \subseteq \text{MHP}_{\text{sem}}(s)$ as desired.

Lemmas 9 and 16 give the following result.

Theorem 10. $\text{MHP}_{\text{sem}}(s) = \text{MHP}_{\text{type}}(s)$.

We can compute $\text{MHP}_{\text{type}}(s)$ in $O(n^3)$ time. To see that, notice that the type rules define a recursive procedure which does a single pass over the program. During that pass, we will have to do $O(n)$ union operations on sets that are of size $O(n^2)$, namely the union operations $M_1 \cup M_2 \cup \text{symcross}(O_1, L_2)$ in Rule (8), and $M \cup \text{symcross}(O, L)$ in Rule (9). We conclude that the total running time is $O(n^3)$.

When we combine Theorem 10 and the $O(n^3)$ time procedure for computing $\text{MHP}_{\text{type}}(s)$, we get that we can solve the MHP computation problem for programs without procedures in $O(n^3)$ time. This is consistent with Theorem 2,

although the algorithm based on Theorem 2 is strictly more powerful because it can handle procedures. However, the constant factor in the linear algorithm is large, and the algorithm is not easy to implement. In contrast, the $O(n^3)$ time procedure for computing $\text{MHP}_{\text{type}(s)}$ is straightforward and easy to implement.

7 Conclusion

We have presented new complexity results for may-happen-in-parallel analysis for X10 programs, and in particular we have presented a practical recursive procedure for programs without procedures that can be extended to give conservative, approximate answers for program with procedures. Our results show that the may-happen-in-parallel analysis problem for languages with async-finish parallelism is computationally tractable, as opposed to the situation for concurrent languages with rendezvous or locks.

We have implemented our type-based algorithm for MHP analysis of X10 programs and we have run it on 13 benchmarks taken from the HPC challenge benchmarks, the Java Grande benchmarks converted into X10, the NAS benchmarks (see [9] for details on the benchmarks), and two benchmarks we wrote ourselves. In total we have over 15K lines of code. Our largest benchmark, plasma consists of over 4500 lines of code with 151 asyncs and 84 finishes. Our implementation additionally uses a conservative rule to handle potentially recursive procedures. The time taken for our experiments was mostly less than a second, with the largest benchmark (plasma) finishing in 4s. The times were somewhat faster than reported for the static analysis in [9]. Our experimental results demonstrate that our algorithm scales to realistic benchmarks.

References

1. Shivali Agarwal, Rajkishore Barik, Vivek Sarkar, and R. K. Shyamasundar. May-happen-in-parallel analysis of X10 programs. In *PPOPP*, pages 183–193. ACM, 2007.
2. Alfred V. Aho, Ravi I. Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, second edition, 1986.
3. Rajkishore Barik. Efficient computation of may-happen-in-parallel information for concurrent Java programs. In *LCPC*, pages 152–169, 2005.
4. Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR*, pages 473–487, 2005.
5. Philippe Charles, Christopher Donawa, Kemal Ebcioglu, Christian Grothoff, Allan Kielstra, Vivek Sarkar, and Christoph Von Praun. X10: An object-oriented approach to non-uniform cluster computing. In *OOPSLA 05*, pages 519–538. ACM SIGPLAN, 2005.
6. Jong-Deok Choi, Keunwoo Lee, Alexey Loginov, Robert O’Callahan, Vivek Sarkar, and Manu Sridharan. Efficient and precise datarace detection for multithreaded object-oriented programs. In *PLDI*, pages 258–269, 2002.
7. Evelyn Duesterwald and Mary Lou Soffa. Concurrency analysis in the presence of procedures using a data-flow framework. In *Symposium on Testing, Analysis, and Verification*, pages 36–48, 1991.

8. Vineet Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise CFL-reachability for threads communicating via locks. In *LICS'09*, pages 27–36, 2009.
9. Jonathan K. Lee and Jens Palsberg. Featherweight X10: a core calculus for asynchronous parallelism. In *PPOPP'10*, 2010.
10. Lin Li and Clark Verbrugge. A practical MHP information analysis for concurrent Java programs. In *LCPC*, pages 194–208, 2004.
11. Stephen P. Masticola and Barbara G. Ryder. Non-concurrency analysis. In *PPOPP*, pages 129–138, 1993.
12. Mayur Naik and Alex Aiken. Conditional must not aliasing for static race detection. In *Proceedings of POPL'07, SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 327–338, 2007.
13. Gleb Naumovich and George S. Avrunin. A conservative data flow algorithm for detecting all pairs of statement that may happen in parallel. In *SIGSOFT FSE*, pages 24–34, 1998.
14. Gleb Naumovich, George S. Avrunin, and Lori A. Clarke. An efficient algorithm for computing HP information for concurrent Java programs. In *ESEC / SIGSOFT FSE*, pages 338–354, 1999.
15. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Transactions on Programming Languages and Systems*, 22(2):416–430, 2000.
16. Richard N. Taylor. Complexity of analyzing the synchronization structure of concurrent programs. *Acta Inf.*, 19:57–84, 1983.