Systematic Realisation of Control Flow Analyses for CML

Kirsten L. Solberg Gasser, Flemming Nielson, Hanne Riis Nielson

Computer Science Department, Aarhus University,
Ny Munkegade, DK-8000 Aarhus C, Denmark

Electronic mail: {kls,fn,hrn}@daimi.aau.dk

Abstract We present a methodology for the systematic realisation of control flow analyses and illustrate it for Concurrent ML. We start with an abstract *specification* of the analysis that is next proved semantically sound with respect to a traditional small-step operational semantics; this result holds for terminating as well as non-terminating programs. The analysis is defined coinductively and it is shown that all programs have a least analysis result (that is indeed the best one). To *realise* the analysis we massage the specification in three stages: (i) to explicitly record reachability of subexpressions, (ii) to be defined in a syntax-directed manner, and (iii) to generate a set of constraints that subsequently can be solved by standard techniques. We prove equivalence results between the different versions of the analysis; in particular it follows that the least solution to the constraints generated will be the least analysis result also to the initial specification.

1 Introduction

Many compiler optimisation techniques rely on control flow information: for a given program point, which program points can the flow of control jump to? For imperative programs without procedures this question may be simple to answer but for more powerful languages, whether imperative languages with procedures, functional languages, concurrent languages, or object-oriented languages, it is much more complicated. In the literature quite some effort has been devoted to the development of control flow analyses for functional languages, see e.g. [1, 5, 8, 9, 10, 13, 15, 18, 19, 20, 21].

We believe that the overall development of control flow analyses should follow the *core methodology* of abstract interpretation [2, 3, 4, 11]:

$$sem \rightsquigarrow pa_0 \longrightarrow \cdots \longrightarrow pa_n \rightsquigarrow impl$$

Given a programming language and its semantics (sem), we consider an abstract analysis (pa_0) and show it to be semantically correct. We then systematically massage the analyses in such a way that (i) semantic correctness is maintained, and (ii) we end up with an analysis (pa_n) that may be implemented efficiently ($impl$). The core methodology also opens up for the possibility of coarsening the analyses (using Galois connections, widening operators, and narrowing operators) but we are not going to explore this in the present paper.

We shall illustrate the methodology by developing a control flow analysis for Concurrent ML (CML) [17]. CML is an extension of the functional language Standard ML (SML) [12] with primitives for the dynamic creation of processes and channels and for the synchronous communication of values over channels. Channels as well as functions are first class values and thus they can be supplied to functions as parameters, returned as results, and transmitted over channels. Compared with traditional functional languages, the control flow of CML programs is further complicated by the fact that

- closures may be communicated over channels and hence invoked on other processes than where they are created, and
- channels may be communicated over channels and later used for new communications.

The analysis we present in this paper is an extension of the traditional 0-CFA analyses for functional languages to the concurrency primitives of CML.

Relationship to other work In the literature several control flow analyses have been developed for functional languages; some using the standard syntax for such languages and others using an intermediate language (like continuation passing style or "A-normal form") of a specific implementation. The analyses have been specified in different styles and even if we restrict attention to those lending themselves naturally to a constraint based formalisation the variation is surprisingly large. In one end of the spectrum the specifications are rather abstract: constraints are only specified implicitly and subexpressions may be analysed several times depending on the context in which they arise. In the other end of the spectrum we have analyses that are specified at a level much closer to the actual realisation: they are explicit about the generation of constraints and they proceed in a syntax-directed manner such that subexpressions are analysed at most once. Only very recently [13] it was pointed out that the first kind of analyses should be defined *coinductively* (and not inductively) in order to make sense; the second kind of analyses are naturally defined *inductively*. We believe the present paper is the first to explore the *formal relationship* between the two styles of specification.

Control flow analysis of concurrent languages have only received rather little attention: Jagannathan and Weeks [8] study a lambda calculus extended with a `spawn` construct

for creating parallel threads that communicate via first class shared locations, and they present a 0-CFA like analysis with the additional twist that a 1-CFA like approach is used to distinguish processes. Flanagen and Felleisen [6] study a lambda calculus with a `future` construct for creating parallel threads and they present a set-based analysis [7]; the precision of the analysis corresponds to that of a 0-CFA analysis. Nielson and Nielson [14] study a subset of CML and present an analysis that predicts the communication topology of programs expressed as terms in a process algebra; the analysis is presented as a type and effect system and is primarily concerned with the prediction of the (annotated) types of the various program fragments.

Overview In Section 2 we present the syntax and semantics of a subset of CML. The syntax is the standard one [17]; here we deviate from much of the literature in that we do not require programs to be written in some intermediate form like "A-normal form" or continuation passing style. The semantics is a traditional small-step operational semantics using environments [16]; this means that we do not require the semantics to operate on contours or to be at the level of an abstract machine.

The *abstract closure analysis* is specified in Section 3. It is defined coinductively as the *greatest fixed point* of a certain function and specifies all acceptable analyses of the programs. Obviously this includes the least acceptable analysis but the present approach opens up for the possibility of having the implementation compute a more approximate (and hence cheaper) solution; this is indeed a scenario common in abstract interpretation.

The correctness of the analysis is formulated as a *subject reduction result*; this is an approach borrowed from type theory. The correctness result is *not* restricted to terminating programs and this is important not least for concurrent languages where non-terminating programs (as operating systems) are of interest on their own. We also show that the analysis enjoys a *Moore family property* (sometimes called a model intersection property); this means that all programs can be analysed and that all programs have a least analysis result (which is indeed the best one).

The next step towards realisation of the analysis is to turn it into a syntax-directed specification. To do that we first note that the abstract analysis may analyse the same program fragments several times but (unlike type systems) it is only concerned with the reachable program fragments. As a first step towards the syntax-directed analysis, Section 4 reformulates the abstract analysis to *explicitly* keep track of the *reachable* program fragments although they may still be analysed several times. The resulting *reachability closure analysis* is also defined as the greatest fixed point of a function, and a coinductive proof shows that it admits the same analyses as the abstract closure analysis of Section 3, and that it enjoys a Moore family property. This means that the least analysis result of the modified analysis also is the least analysis result of the original one and vice versa.

In Section 5 we present the *syntax-directed closure analysis*. It incorporates the reachability aspect of the previous analysis but in such a way that each program fragment is analysed *at most once*. Since this analysis is syntax-directed we can safely define it as the least fixed point of the specification as indeed it will only have one fixed point. We show that the analysis results obtainable by the syntax-directed specification also are obtainable in the reachability specification and furthermore, that the *least* (or best) analysis result that can be obtained by the reachability specification equals the one obtained by the syntax-directed specification.

The syntax-directed specification can then be turned into a syntax-directed function $\mathcal{C}$ for collecting constraint. We illustrate this in Section 6 where we also show that the constraints generated have the same solutions as the syntax-directed specification. Standard constraint solving algorithms can now be employed to find the least solution to the constraints, and we briefly sketch an $O(n^3)$ algorithm for this (where n is the size of the program). Composing the correctness results then yields an implementation that will compute the least analysis result with respect to the abstract specification of the analysis and we know that it will be semantically correct.

Finally, Section 7 contains the concluding remarks. The Appendix contains the non-trivial parts of the proofs of the main results of the paper but can safely be omitted; it also contains details of the constraint solving technique.

2 Concurrent ML

Concurrent ML is an extension of SML with concurrency primitives for synchronous communication over channels. We shall be interested in the following operations:

"`fork` e" creates a new process; the expression e must evaluate to a function expecting a unit argument and the process will execute the expression e `()`.

"`channel` e" creates a new channel; the expression e must evaluate to the value `()`.

"`send` e_1 e_2" transmits the value of e_2 on the channel that e_1 evaluates to (provided that another process will engage in the communication).

"`receive` e" accepts a value on the channel that e evaluates to (provided that another process will engage in the communication).

We shall consider a subset of CML that additionally has explicit operators for conditional, recursion and local definitions. Note that `send` and `receive` take care of their own synchronisation; hence we can dispense with the `sync` operation. For the presentation of our analysis it is important that we are able to *label* all program fragments and we therefore present the syntax using expressions and terms:

$e \in Exp$ (expressions, i.e. labelled terms)
$e ::= t^l$

$t \in Term$ (terms, i.e. unlabelled expressions)
$t ::= c \mid x \mid$ `fn` x `=>` $e_0 \mid$ `fun` f x `=>` e_0
$\mid e_1\ e_2 \mid$ `if` e_0 `then` e_1 `else` e_2
$\mid$ `let` x `=` e_1 `in` $e_2 \mid$ `fork` $e \mid$ `channel` e
$\mid$ `send` $e_1\ e_2 \mid$ `receive` e

$l \in Lab$ (labels)
$c \in Const$ (constants)
$x \in Var$ (variables)

Here `fn` x `=>` e is the function abstraction, `fun` f x `=>` e is a recursive variant of `fn` x `=>` e where all free occurrences of f in e refer to `fun` f x `=>` e itself, and `let` x `=` e_1 `in` e_2 is a non-recursive local definition that is equivalent to (`fn`

x `=>` $e_2)(e_1)$. As usual we shall use parentheses to disambiguate the parsing whenever needed. Also we shall *assume* throughout that all occurrences of `fun` f x `=>` e have f and x to be distinct. For simplicity of presentation we shall *assume* that there are no functional constants; however, it would be straightforward to add a syntactic clause like "$t ::= e_1\ op\ e_2$" for a class *op* of binary operators (like addition).

We shall equip this language with a small-step operational semantics using environments in the style proposed by Plotkin [16]. This necessitates augmenting the syntax with notation for closures and for local environments. We therefore define *intermediate* expressions and terms as follows:

$$
\begin{array}{ll}
ie \in \mathit{IExp} & \text{(intermediate expressions)} \\
ie ::= it^l & \\
it \in \mathit{ITerm} & \text{(intermediate terms)} \\
it ::= c \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ e_0 \mid \texttt{fun}\ f\ x\ \texttt{=>}\ e_0 & \\
\quad \mid ie_1\ ie_2 \mid \texttt{if}\ ie_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \\
\quad \mid \texttt{let}\ x\ \texttt{=}\ ie_1\ \texttt{in}\ e_2 \mid \texttt{fork}\ ie & \\
\quad \mid \texttt{channel}\ ie \mid \texttt{send}\ ie_1\ ie_2 \mid \texttt{receive}\ ie & \\
\quad \mid ch \mid \texttt{bind}\ \rho\ \texttt{in}\ ie \mid \texttt{close}\ t\ \texttt{in}\ \rho & \\
v \in \mathit{Val} & \text{(values)} \\
v ::= c \mid ch \mid \texttt{close}\ t\ \texttt{in}\ \rho & \\
\rho \in \mathit{Env} & \text{(environments)} \\
\rho ::= [\,] \mid \rho[x \mapsto v] & \\
ch \in \mathit{ChId} & \text{(channel identifiers)}
\end{array}
$$

The "top-level" semantics of the *sequential* subset of the language is given by the transition system $\rho \vdash ie \to ie'$ of Table 1. To overcome the restriction to "top-level" we shall make use of evaluation contexts E:

$$
\begin{array}{l}
E ::= [\,] \mid (E\ e_2)^l \mid (v\ E)^l \mid (\texttt{if}\ E\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^l \\
\quad \mid (\texttt{let}\ x\ \texttt{=}\ E\ \texttt{in}\ e_2)^l \mid (\texttt{fork}\ E)^l \mid (\texttt{channel}\ E)^l \\
\quad \mid (\texttt{send}\ E\ e_2)^l \mid (\texttt{send}\ v\ E)^l \mid (\texttt{receive}\ E)^l \\
\quad \mid (\texttt{bind}\ \rho\ \texttt{in}\ E)^l
\end{array}
$$

We have been very deliberate in when to use intermediate expressions and when to use expressions. Since we do not evaluate the body of a function before it is applied we continue to let the body be an expression rather than an intermediate expression. Similar remarks apply to the branches of the conditional and the body of the local definitions. Note that although an environment only records the terms `fn` x `=>` e_0 and `fun` f x `=>` e_0 occurring in the closures, we do not lose the identity of the function abstractions as e_0 will be of the form $t_0^{l_0}$ and l_0 may be used to "uniquely" identify the function abstraction.

In order not to lose the identity of the channel identifiers we shall introduce a finitary function

$$K : \mathit{ChId} \to \mathit{Term}$$

that specifies which occurrence of `channel` e that gives rise to the channel identifier. The *concurrent semantics* is given by the transition system $K, PP \longrightarrow K', PP'$ of Table 1. Configurations have the form K, PP where K is as above and PP is a finitary mapping from process identifiers $p \in \mathit{PId}$ to intermediate expressions; these intermediate expressions will always be *closed* (having no free variables) but they may contain channel identifiers that are present in the domain of K.

In order to evaluate inside a `bind`-construct we have to reconstruct the local environment. For this we use the function $\mathcal{E}$ defined by:

$$
\begin{array}{rcl}
\mathcal{E}(\rho, [\,]) & = & \rho \\
\mathcal{E}(\rho, (E\ e_2)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (v\ E)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{if}\ E\ \texttt{then}\ e_1\ \texttt{else}\ e_2)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{let}\ x\ \texttt{=}\ E\ \texttt{in}\ e_2)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{fork}\ E)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{channel}\ E)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{send}\ E\ e_2)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{send}\ v\ E)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{receive}\ E)^l) & = & \mathcal{E}(\rho, E) \\
\mathcal{E}(\rho, (\texttt{bind}\ \rho_1\ \texttt{in}\ E)^l) & = & \mathcal{E}(\rho_1, E)
\end{array}
$$

Remark One might have expected the rule for the `fork`-construct to look like:

$$
\begin{array}{l}
K, PP[p : E[(\texttt{fork}\ \ (\texttt{close}\ t\ \texttt{in}\ \rho)^{l_1})^l]] \longrightarrow \\
\qquad K, PP[p : E[()^l]]\ [p' : (t^?\ \ ()^?)^?] \\
\qquad\qquad \text{if } p' \notin \mathit{dom}(PP) \cup \{p\}
\end{array}
$$

However, with this approach it is unclear what labels to put instead of the questions-marks.

3 Abstract Closure Analysis

The result of the 0-CFA analysis for CML is a triple $(\widehat{C}, \widehat{\rho}, \widehat{\kappa})$ where

- $\widehat{C}$ is the *abstract cache* associating an abstract value with each labelled program point; this denotes the set of values that the program point could evaluate to;
- $\widehat{\rho}$ is the *abstract environment* associating an abstract value with each variable; this denotes the set of values that the variable could be bound to; and
- $\widehat{\kappa}$ is the *abstract channel environment* which associates an abstract value with each labelled program point denoting a channel; this denotes the set of values that could be transmitted over a channel created at that program point.

This is made precise by the following definitions:

$$
\begin{array}{rcl}
\widehat{v} \in \widehat{\mathit{Val}} & = & \mathcal{P}(\mathit{Term}) \\
\widehat{C} \in \widehat{\mathit{Cache}} & = & \mathit{Lab} \to \widehat{\mathit{Val}} \\
\widehat{\rho} \in \widehat{\mathit{Env}} & = & \mathit{Var} \to \widehat{\mathit{Val}} \\
\widehat{\kappa} \in \widehat{\mathit{KEnv}} & = & \mathit{Lab} \to \widehat{\mathit{Val}}
\end{array}
$$

Here an abstract value $\widehat{v}$ only records a set of terms of form `fn` x `=>` e, `fun` f x `=>` e, or `channel` $()^l$. It does not record abstract versions of the corresponding local environments as these are collapsed into the global environment $\widehat{\rho}$ in 0-CFA analyses; also it means that the analysis will be unable to record any form of causality. So for functions we record the corresponding abstraction in the program as usual and for channels we record the occurrence of `channel` t^l that gives

(var) $\rho \vdash x^l \to v^l$ if $\rho(x) = v$

(fn) $\rho \vdash (\texttt{fn}\ x\ \texttt{=>}\ e)^l \to (\texttt{close}\ (\texttt{fn}\ x\ \texttt{=>}\ e)\ \texttt{in}\ \rho)^l$

(fun) $\rho \vdash (\texttt{fun}\ f\ x\ \texttt{=>}\ e)^l \to (\texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e)\ \texttt{in}\ \rho)^l$

(app_{fn}) $\rho \vdash ((\texttt{close}\ (\texttt{fn}\ x\ \texttt{=>}\ e)\ \texttt{in}\ \rho_1)^{l_1}\ v^{l_2})^l \to (\texttt{bind}\ \rho_1[x \mapsto v]\ \texttt{in}\ e)^l$

(app_{fun}) $\rho \vdash ((\texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e)\ \texttt{in}\ \rho_1)^{l_1}\ v^{l_2})^l \to (\texttt{bind}\ \rho_2[x \mapsto v]\ \texttt{in}\ e)^l$
if $\rho_2 = \rho_1[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e)\ \texttt{in}\ \rho_1]$

(let) $\rho \vdash (\texttt{let}\ x\ \texttt{=}\ v^{l_1}\ \texttt{in}\ e)^l \to (\texttt{bind}\ \rho[x \mapsto v]\ \texttt{in}\ e)^l$

(ifT) $\rho \vdash (\texttt{if}\ \texttt{true}^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l \to t_1^l$

(ifF) $\rho \vdash (\texttt{if}\ \texttt{false}^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l \to t_2^l$

(bind) $\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ v^{l_1})^l \to v^l$

(seq) $K, PP[p : E[e_1]] \longrightarrow K, PP[p : E[e_2]]$ if $\mathcal{E}(\emptyset, E) \vdash e_1 \to e_2$

(chan) $K, PP[p : E[(\texttt{channel}\ ()^{l_1})^l]] \longrightarrow K[ch \mapsto \texttt{channel}\ ()^{l_1}], PP[p : E[ch^l]]$ if $ch \notin dom(K)$

(fork_{fn}) $K, PP[p : E[(\texttt{fork}\ (\texttt{close}\ (\texttt{fn}\ x\ \texttt{=>}\ t^{l_2})\ \texttt{in}\ \rho)^{l_1})^l]] \longrightarrow$
$K, PP\begin{bmatrix}p : E[()^l]] \\ p' : (\texttt{bind}\ \rho[x \mapsto ()]\ \texttt{in}\ t^{l_2})^{l_2}]\end{bmatrix}$ if $p' \notin dom(PP) \cup \{p\}$

(fork_{fun}) $K, PP[p : E[(\texttt{fork}\ (\texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ t^{l_2})\ \texttt{in}\ \rho)^{l_1})^l]] \longrightarrow$
$K, PP\begin{bmatrix}p : E[()^l]] \\ p' : (\texttt{bind}\ \rho[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ t^{l_2})\ \texttt{in}\ \rho][x \mapsto ()]\ \texttt{in}\ t^{l_2})^{l_2}]\end{bmatrix}$
if $p' \notin dom(PP) \cup \{p\}$

(comm) $K, PP\begin{bmatrix}p_1 : E_1[(\texttt{send}\ ch^{l_1}\ v^{l_2})^l]] \\ p_2 : E_2[(\texttt{receive}\ ch^{l_3})^{l_4}]]\end{bmatrix} \longrightarrow K, PP\begin{bmatrix}p_1 : E_1[v^l]] \\ p_2 : E_2[v^{l_4}]]\end{bmatrix}$ if $p_1 \neq p_2$

Table 1: Small-Step Semantics

rise to the channel. However, t is not stable under evaluation and therefore we decide to use its final value `()` instead. We will not be recording any constants among the abstract values and we thus obtain a "pure" control flow analysis with no data flow analysis component. As previously mentioned it is straightforward to extend the development to record simple properties of simple data structures (like detection of signs for integers); algebraic data structures requires a little more work as does the use of infinite sets of properties. We do not need to assume that all bound variables are distinct but clearly greater precision is achieved if this is the case; parts of the development will need to assume that all labels are distinct (see Section 5).

We shall shortly formulate the correctness of the analysis as a subject reduction property and this means that we will need to analyse a pool PP of processes. Each process identifier p is mapped to an intermediate expression so we will also have to specify the analysis for the *intermediate terms* ch, `close` t `in` ρ, and `bind` ρ `in` e. The general formulation of the 0-CFA analysis will therefore have the form

$$(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ie$$

and it expresses that $(\widehat{C}, \widehat{\rho}, \widehat{\kappa})$ is an acceptable closure analysis of the intermediate expression ie where the channel identifiers are mapped to their syntactic creation points by the function K. The analysis of expressions is specified in Table 2 and it is extended to an analysis of a process pool by

$$\begin{array}{l}(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K PP \text{ iff} \\ \forall p \in dom(PP) : (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K PP(p)\end{array}$$

The clauses of Table 2 contain a number of inclusions of the form

$$lhs \subseteq rhs$$

where rhs is of the form $\widehat{C}(l)$, $\widehat{\rho}(x)$ or $\widehat{\kappa}(l)$ and where lhs is of the form $\widehat{C}(l), \widehat{\rho}(x), \widehat{\kappa}(l)$, or $\{t\}$. These inclusions express how the higher-order entities may flow through the program: actual parameters flow into formal parameters, results of function calls flow back to the call sites, etc.

A number of clauses also contain "recursive calls" to the acceptability relation ($\models_K$). The noteworthy exceptions to this pattern are the clauses (fn), (fun), and (close) that in-

(const) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K c^l$ iff $true$

(var) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K x^l$ iff $\widehat{\rho}(x) \subseteq \widehat{C}(l)$

(fn) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{fn}\ x\ \texttt{=>}\ e)^l$ iff $(\texttt{fn}\ x\ \texttt{=>}\ e) \in \widehat{C}(l)$

(fun) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{fun}\ f\ x\ \texttt{=>}\ e)^l$ iff $(\texttt{fun}\ f\ x\ \texttt{=>}\ e) \in \widehat{C}(l)$

(app) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (it_1^{l_1}\ it_2^{l_2})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_2^{l_2} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_0^{l_0} \wedge$
$\widehat{C}(l_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(l_0) \subseteq \widehat{C}(l)) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_0^{l_0} \wedge$
$\widehat{C}(l_2) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(l_0) \subseteq \widehat{C}(l) \wedge$
$(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f))$

(if) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{if}\ ie\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ie \wedge (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_1^{l_1} \wedge (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_2^{l_2} \wedge$
$\widehat{C}(l_1) \subseteq \widehat{C}(l) \wedge \widehat{C}(l_2) \subseteq \widehat{C}(l)$

(let) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{let}\ x\ \texttt{=}\ it_1^{l_1}\ \texttt{in}\ t_2^{l_2})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_2^{l_2} \wedge$
$\widehat{C}(l_1) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(l_2) \subseteq \widehat{C}(l)$

(fork) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{fork}\ it_1^{l_1})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_0^{l_0}) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K t_0^{l_0} \wedge (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f))$

(chan) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{channel}\ it_1^{l_1})^l$ iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge (\texttt{channel}\ ()^{l_1}) \in \widehat{C}(l)$

(send) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{send}\ it_1^{l_1}\ it_2^{l_2})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_2^{l_2} \wedge$
$\widehat{C}(l_2) \subseteq \widehat{C}(l) \wedge \forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : \widehat{C}(l_2) \subseteq \widehat{\kappa}(l_0)$

(receive) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{receive}\ it_1^{l_1})^l$
iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge$
$\forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : \widehat{\kappa}(l_0) \subseteq \widehat{C}(l)$

(ch) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ch^l$ iff $K(ch) \in \widehat{C}(l)$

(close) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{close}\ t\ \texttt{in}\ \rho)^l$ iff $t \in \widehat{C}(l) \wedge \rho\ \mathcal{R}_K\ \widehat{\rho}$

(bind) $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K (\texttt{bind}\ \rho\ \texttt{in}\ it_1^{l_1})^l$ iff $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_1^{l_1} \wedge \widehat{C}(l_1) \subseteq \widehat{C}(l) \wedge \rho\ \mathcal{R}_K\ \widehat{\rho}$

Table 2: Abstract Closure Analysis ($\models_K$)

stead rely on (app) and (fork) to perform the "recursive calls" on all closures that are possibly applied. As a consequence some expressions will be analysed more than once since they may be applied at different program points whereas other will not be analysed at all since they will not be reachable. The latter is a phenomenon common in program analysis, where there is no need to analyse unreachable fragments, but is different from the perspective of type inference, where even unreachable fragments must be correctly typed.

To explain the clauses (close) and (bind) we shall first introduce the correctness relation $\mathcal{R}_K$ that expresses when an environment of the semantics is correctly modelled by an abstract environment of the analysis:

$$\begin{array}{rll} \rho\ \mathcal{R}_K\ \widehat{\rho} & \text{iff} & \forall x \in dom(\rho) \subseteq dom(\widehat{\rho}) : \\ & & \rho(x)\ \mathcal{V}_K\ (\widehat{\rho}, \widehat{\rho}(x)) \\ v\ \mathcal{V}_K\ (\widehat{\rho}, \widehat{v}) & \text{iff} & (\forall t, \rho : (v = (\texttt{close}\ t\ \texttt{in}\ \rho)) \Rightarrow \\ & & \quad (t \in \widehat{v}) \wedge (\rho\ \mathcal{R}_K\ \widehat{\rho})) \wedge \\ & & (\forall ch : (v = ch) \Rightarrow (K(ch) \in \widehat{v})) \end{array}$$

These relations are defined mutually recursively in terms of one another. Note that in the definition of $\mathcal{V}_K$ the local environment ρ is related to the global abstract environment $\widehat{\rho}$ as the abstract value $\widehat{v}$ does not contain an abstract environment. Also note that the semantic entities (values v and environments ρ) decrease in size as we perform the "recursive calls"; thus a simple well-founded induction in the finite size of the semantic entities suffices for showing well-definedness of these relations.

Finally, we need to clarify that the clauses of Table 2 indeed define a relation. The difficulty here is that the clauses for (app) and (fork) are not of a form that allows to define $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_K ie$ by structural induction in the intermediate expression ie. Instead we note [13] that all right hand sides are monotone in "$\models$" and hence we can define $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_K ie$ as the *greatest* fixed point of the above specification; this is often called a *coinductive* definition.

Properties of the Analysis

We shall formulate semantic correctness of the analysis as a subject reduction result:

Theorem 1 *Semantic Correctness*
If $K, PP \longrightarrow K', PP'$ and $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_K PP$ then $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_{K'} PP'$. □

Thus the analysis results obtained for the initial configuration will also hold for all subsequent configurations.

Having defined the analysis in Table 2 it is natural to ask for each program, whether or not it admits a closure analysis and if so, whether or not there is a least (or best) closure analysis. Here least is defined with respect to the partial order

$$\begin{array}{lll}(\widehat{C}_1,\ \widehat{\rho}_1,\ \widehat{\kappa}_1) \sqsubseteq (\widehat{C}_2,\ \widehat{\rho}_2,\ \widehat{\kappa}_2) & \text{iff} & (\forall l : \widehat{C}_1(l) \subseteq \widehat{C}_2(l)) \wedge \\ & & (\forall x : \widehat{\rho}_1(x) \subseteq \widehat{\rho}_2(x)) \wedge \\ & & (\forall l : \widehat{\kappa}_1(l) \subseteq \widehat{\kappa}_2(l))\end{array}$$

We shall show that the answers to both questions are yes. In fact we can show that the least analysis of a program only "mentions" terms appearing in the program.

To show these results let PP_* be a finitary mapping from PId to closed expressions in Exp; this will be the process pool of main interest throughout the rest of the paper. Let $Lab_* \subseteq Lab$ be the finite set of labels occurring in the range of PP_*; let $Var_* \subseteq Var$ be the finite set of variables occurring in the range of PP_*; let $Term_* \subseteq Term$ be the finite set of subterms occurring in the range of PP_* together with all terms $\texttt{channel}\ ()^l$ such that some $\texttt{channel}\ t^l$ occurs in the range of PP_*; and let $Exp_* \subseteq Exp$ be the finite set of subexpressions occurring in the range of PP_*. Now define $(\widehat{C_*^\top},\ \widehat{\rho_*^\top},\ \widehat{\kappa_*^\top})$ by:

$$\widehat{C_*^\top}(l) = \begin{cases} \emptyset & \text{if } l \notin Lab_* \\ Term_* & \text{if } l \in Lab_* \end{cases}$$

$$\widehat{\rho_*^\top}(x) = \begin{cases} \emptyset & \text{if } x \notin Var_* \\ Term_* & \text{if } x \in Var_* \end{cases}$$

$$\widehat{\kappa_*^\top}(l) = \begin{cases} \emptyset & \text{if } l \notin Lab_* \\ Term_* & \text{if } l \in Lab_* \end{cases}$$

Note that the claim $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \sqsubseteq (\widehat{C_*^\top},\widehat{\rho_*^\top},\widehat{\kappa_*^\top})$ is for all practical purposes equivalent to the claim $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \in \widehat{Cache}_* \times \widehat{Env}_* \times \widehat{KEnv}_*$ where $\widehat{Cache}_* = Lab_* \to \widehat{Val}_*$, $\widehat{Env}_* = Var_* \to \widehat{Val}_*$, $\widehat{KEnv}_* = Lab_* \to \widehat{Val}_*$ and $\widehat{Val}_* = \mathcal{P}(Term_*)$. Thus the condition $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \sqsubseteq (\widehat{C_*^\top},\widehat{\rho_*^\top},\widehat{\kappa_*^\top})$ expresses that only terms "occurring" in PP_* are present in the range of $\widehat{C}$, $\widehat{\rho}$ and $\widehat{\kappa}$.

Theorem 2 *Moore Family Property*
Both of the sets $\{(\widehat{C},\widehat{\rho},\widehat{\kappa}) \mid (\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_K PP_*\}$ and $\{(\widehat{C},\widehat{\rho},\widehat{\kappa}) \sqsubseteq (\widehat{C_*^\top},\widehat{\rho_*^\top},\widehat{\kappa_*^\top}) \mid (\widehat{C},\widehat{\rho},\widehat{\kappa}) \models_K PP_*\}$ are Moore families and are independent of K; their greatest elements are $(\lambda l.Term, \lambda x.Term, \lambda l.Term)$ and $(\widehat{C_*^\top},\widehat{\rho_*^\top},\widehat{\kappa_*^\top})$, respectively; also they have the same least element. □

This result shows that for the purpose of finding (least) solutions it suffices to restrict $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \in \widehat{Cache} \times \widehat{Env} \times \widehat{KEnv}$ to $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \in \widehat{Cache}_* \times \widehat{Env}_* \times \widehat{KEnv}_*$. To remind us of this we shall allow to write $(\widehat{C},\widehat{\rho},\widehat{\kappa}) \models^* \cdots$ and from now on we shall *assume* that $(\widehat{C},\widehat{\rho},\widehat{\kappa})$ is restricted in this manner.

One can prove that the Moore family property would fail to hold for an inductive rather than coinductive definition of $\models_K$.

4 Reachability

The aim now is to massage the specification of the analysis of the previous section so as to make it amenable for efficient implementation. It follows that we are no longer interested in the analysis of intermediate expressions; these were only necessary for establishing the semantic correctness result. Henceforth we will need no counterparts of the clauses (ch), (close) and (bind) in Table 2 and therefore we can dispense with the K-component.

We will be aiming for a syntax-directed procedure for collecting constraints such that

- only expressions that are reachable will actually be analysed, and
- the same expression is analysed at most once.

We shall proceed in three stages: First we modify the analysis of Table 2 to track the reachability of subexpressions explicitly (Section 4), then we modify it such that subexpressions are analysed at most once (Section 5), and finally we introduce the syntax-directed procedure for collecting constraints (Section 6).

We shall extend the analysis of Table 2 to track the reachability of the subexpressions *explicitly*. To this end we define

$$\widehat{R} \in \widehat{Reach}_* = Lab_* \to \mathcal{P}(\{\texttt{on}\})$$

The idea is that if $\widehat{R}(l) = \emptyset$ then the program point l is not reachable and hence the corresponding expression will not be analysed. For this to work (in Section 5) we shall demand that all expressions in PP_* are uniquely labelled (but we still do not demand that variables are unique).

(const) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} c^{l}$ iff $true$

(var) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} x^{l}$ iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{\rho}(x) \subseteq \widehat{C}(l))$

(fn) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{fn}\ x\ \texttt{=>}\ e)^{l}$ iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{fn}\ x\ \texttt{=>}\ e) \in \widehat{C}(l))$

(fun) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{fun}\ f\ x\ \texttt{=>}\ e)^{l}$ iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ e) \in \widehat{C}(l))$

(app) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (t_1^{l_1}\ t_2^{l_2})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_2^{l_2} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{l_0} t_0^{l_0} \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)) \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\rho}(x)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_0) \subseteq \widehat{C}(l))) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{l_0} t_0^{l_0} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\rho}(x)) \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_0) \subseteq \widehat{C}(l)) \wedge (\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f)))$

(if) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{if}\ e\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} e \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_1) \subseteq \widehat{C}(l)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l))$

(let) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{let}\ x\ \texttt{=}\ t_1^{l_1}\ \texttt{in}\ t_2^{l_2})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_1) \subseteq \widehat{\rho}(x)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l))$

(fork) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{fork}\ t_1^{l_1})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{l_0} t_0^{l_0} \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0))) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{l_0} t_0^{l_0} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)) \wedge (\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f)))$

(chan) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{channel}\ t_1^{l_1})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{channel}\ ()^{l_1}) \in \widehat{C}(l))$

(send) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{send}\ t_1^{l_1}\ t_2^{l_2})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l)) \wedge$
$\forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\kappa}(l_0))$

(receive) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} (\texttt{receive}\ t_1^{l_1})^{l}$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} t_1^{l_1} \wedge$
$\forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{\kappa}(l_0) \subseteq \widehat{C}(l))$

Table 3: Reachability Closure Analysis ($\models^{\ell}$)

The judgements of the previous analysis are now extended to additionally determine the reachability component:

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{\ell} e$$

The modified clauses are given in Table 3. The idea is here that $\ell \in Lab$ is the program point "of interest"; if this program point is reachable according to $\widehat{R}$ then we have conditions on the flow of values similar to those of the previous analysis. The program point "of interest" is changed whenever we need to analyse an expression that is not guaranteed to be a proper subexpression of the current expression, i.e. when we start analysing the body of a function in a function application or in the `fork`-construct.

It is important to note that for the recursive functions we introduce the condition $(\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f))$ reflecting that the recursive call only will happen if the body of the function is to be analysed. An alternative would have been to replace it with $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f))$; while this is "correct", it turns out to be less usable for obtaining the syntax-directed specification.

The analysis is extended to process pools as follows:

$$
\begin{array}{l}
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models PP \text{ iff} \\
\forall p \in dom(PP) : \forall t, l : PP(p) = t^l \Rightarrow \\
\qquad (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^l t^l \wedge \mathtt{on} \in \widehat{R}(l)
\end{array}
$$

It is useful to define

$$
\widehat{R^{\top}_*}(l) = \begin{cases} \emptyset & \text{if } l \notin Lab_* \\ \{\mathtt{on}\} & \text{if } l \in Lab_* \end{cases}
$$

as we then have

Theorem 3 *Preservation of Solutions*
If $(\widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_K PP_*$ then $(\widehat{R^{\top}_*}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$. If $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^* PP_*$ then $(\widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_K PP_*$. Here PP_* is as above and K is arbitrary. □

In analogy with Theorem 2 we get that also

$$
\{(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \mid (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^* PP_*\}
$$

is a Moore family so in particular the least analysis result obtained using Table 3 will correspond to the least analysis result obtained using Table 2 and vice versa.

5 Syntax-Directed Specification

We shall now reformulate the specification of the closure analysis in Table 3 such that all subexpressions are analysed *at most once*. So the aim will be to use the explicit reachability information to guide when the body of function abstractions have to be analysed and then to proceed in a syntax-directed manner. The judgements have the form

$$
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \;\models^{\ell}_s e
$$

much as in the previous section and the clauses are given in Table 4. Note that the the two function abstractions give rise to "recursive calls" of the analysis and that the clauses for function application and process creation only contain "recursive calls" for proper subexpressions. Consequently the recursive definition of "$\models_s$" has precisely one solution. The component $\widehat{R}$ controls whether or not to impose the appropriate inclusions between the flow information. The analysis is extended to process pools as follows:

$$
\begin{array}{l}
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_s PP \text{ iff} \\
\forall p \in dom(PP) : \forall t, l : PP(p) = t^l \Rightarrow \\
\qquad (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^l_s t^l \wedge \mathtt{on} \in \widehat{R}(l)
\end{array}
$$

The following result says that the analysis of Table 4 admits the same results as that of Table 3:

Theorem 4 *Preservation of Solutions*
Suppose that all labels of PP_* are unique. If $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_s PP_*$ then $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^* PP_*$. If $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})$ is *least* such that $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^* PP_*$ then $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_s PP_*$. □

Technically, this is the hardest of the theorems to prove, because Tables 3 and 4 have different perspectives upon what is "global" and what is "local"; see the Appendix for details. Intuitively, it is the leastness of the $\widehat{R}$ component that is essential.

6 Constraint Formulation

We are now ready to consider efficient ways of finding the least solution $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})$ such that $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_s PP_*$. To do so we first construct a finite set $\mathcal{C}_*[\![PP_*]\!]$ of constraints of the forms

$$
\{u\} \subseteq rhs \tag{1}
$$

$$
\{u_1\} \subseteq rhs_1 \Rightarrow lhs \subseteq rhs \tag{2}
$$

$$
\{u_1\} \subseteq rhs_1 \Rightarrow \{u_2\} \subseteq rhs_2 \Rightarrow lhs \subseteq rhs \tag{3}
$$

where rhs is of the form $\tilde{R}(l)$, $\tilde{C}(l)$, $\tilde{\rho}(x)$ or $\tilde{\kappa}(l)$; lhs is of the form $\tilde{C}(l)$, $\tilde{\rho}(x)$, $\tilde{\kappa}(l)$, or $\{u\}$; and u is of the form $\mathtt{on}$ or t. All occurrences of t are of the form $\mathtt{fn}\ x\ \mathtt{=>}\ e$, $\mathtt{fun}\ f\ x\ \mathtt{=>}\ e$ or $\mathtt{channel}\ ()^l$.

The constraints are obtained by expanding $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models^*_s PP_*$ into a finite set of constraints of the above form and then let $\mathcal{C}_*[\![PP_*]\!]$ be the set of individual conjuncts; at the same time we change all occurrences of "$\widehat{\cdots}$" into "$\tilde{\cdots}$" to avoid confusion: so $\widehat{C}(l)$ will be a set of terms whereas $\tilde{C}(l)$ is a formal symbol and similarly for $\widehat{\rho}(x)$ and $\tilde{\rho}(x)$, $\widehat{\kappa}(l)$ and $\tilde{\kappa}(l)$, and $\widehat{R}(l)$ and $\tilde{R}(l)$.

More precisely the definition of $\mathcal{C}_*[\![PP_*]\!]$ is given by

$$
\begin{array}{rcl}
\mathcal{C}_*[\![PP_*]\!] & = & \bigcup \{\mathcal{C}^l_*[\![t^l]\!] \cup \{\{\mathtt{on}\} \subseteq \tilde{R}(l)\} \\
& & \qquad \mid p \in dom(PP_*) \wedge PP_*(p) = t^l\}
\end{array}
$$

where $\mathcal{C}^l_*[\![t^l]\!]$ is given in Table 5. It makes use of the set $Term_*$ of subterms in order to generate only a finite number of constraints.

If the size of PP_* is n then it might seem that there could be $O(n^2)$ constraints of form (1), $O(n^4)$ constraints of form (2) and $O(n^6)$ constraints of form (3). However, inspection of the definitions of $\mathcal{C}^{\ell}_*[\![t^l]\!]$ and $\mathcal{C}_*[\![PP_*]\!]$ shows that there will be at most $O(n)$ constraints of form (1) and (2) and $O(n^2)$ constraints of form (3).

It is important to stress that while $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_s PP_*$ is a logical formula, $\mathcal{C}_*[\![PP_*]\!]$ is a set of formal symbols. We can turn the latter into a logical formula by first translating the "$\tilde{\cdots}$" symbols into the sets "$\widehat{\cdots}$":

$$
\begin{array}{rcl}
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\{t\}]\!] & = & \{t\} \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\{\mathtt{on}\}]\!] & = & \{\mathtt{on}\} \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\tilde{R}(l)]\!] & = & \widehat{R}(l) \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\tilde{C}(l)]\!] & = & \widehat{C}(l) \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\tilde{\rho}(x)]\!] & = & \widehat{\rho}(x) \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![\tilde{\kappa}(l)]\!] & = & \widehat{\kappa}(l)
\end{array}
$$

Next we define a satisfaction relation $(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c \cdots$ on constraints:

$$
\begin{array}{l}
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs \subseteq rhs) \\
\quad \text{iff } (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![lhs]\!] \subseteq (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa})[\![rhs]\!] \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c ((lhs_1 \subseteq rhs_1) \Rightarrow (lhs \subseteq rhs)) \\
\quad \text{iff } ((\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs_1 \subseteq rhs_1) \\
\qquad \Rightarrow (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs \subseteq rhs)) \\
(\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c ((lhs_1 \subseteq rhs_1) \\
\qquad \Rightarrow (lhs_2 \subseteq rhs_2) \Rightarrow (lhs \subseteq rhs)) \\
\quad \text{iff } ((\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs_1 \subseteq rhs_1) \\
\qquad \wedge (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs_2 \subseteq rhs_2) \\
\qquad \Rightarrow (\widehat{R},\, \widehat{C},\, \widehat{\rho},\, \widehat{\kappa}) \models_c (lhs \subseteq rhs))
\end{array}
$$

(const) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell c^l$ iff $true$

(var) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell x^l$ iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{\rho}(x) \subseteq \widehat{C}(l))$

(fn) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0})^l$
iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l)) \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{l_0} t_0^{l_0}$

(fun) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0})^l$
iff $(\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l)) \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{l_0} t_0^{l_0} \wedge$
$(\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f))$

(app) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (t_1^{l_1}\ t_2^{l_2})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_2^{l_2} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)) \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\rho}(x)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_0) \subseteq \widehat{C}(l))) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)) \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\rho}(x)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_0) \subseteq \widehat{C}(l)))$

(if) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{if}\ e\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell e \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_1) \subseteq \widehat{C}(l)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l))$

(let) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{let}\ x\ \texttt{=}\ t_1^{l_1}\ \texttt{in}\ t_2^{l_2})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_1) \subseteq \widehat{\rho}(x)) \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l))$

(fork) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{fork}\ t_1^{l_1})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0))) \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \texttt{on} \in \widehat{R}(l_0)))$

(chan) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{channel}\ t_1^{l_1})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge (\texttt{on} \in \widehat{R}(\ell) \Rightarrow (\texttt{channel}\ ()^{l_1}) \in \widehat{C}(l))$

(send) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{send}\ t_1^{l_1}\ t_2^{l_2})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_2^{l_2} \wedge$
$(\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{C}(l)) \wedge$
$\forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{C}(l_2) \subseteq \widehat{\kappa}(l_0))$

(receive) $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell (\texttt{receive}\ t_1^{l_1})^l$
iff $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^\ell t_1^{l_1} \wedge$
$\forall(\texttt{channel}\ ()^{l_0}) \in \widehat{C}(l_1) : (\texttt{on} \in \widehat{R}(\ell) \Rightarrow \widehat{\kappa}(l_0) \subseteq \widehat{C}(l))$

Table 4: Syntax-Directed Analysis ($\models_s^\ell$)

This definition can be lifted to sets of constraints by:

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c \mathcal{C} \text{ iff } \forall C \in \mathcal{C} : (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c C$$

We can then show that Tables 4 and 5 have the same solutions:

Theorem 5 *Preservation of Solutions*
$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^* PP_*$ if and only if $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c^* \mathcal{C}_*[\![PP_*]\!]$. □

It follows that the least solution $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ to $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^* PP_*$ equals the least solution to $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c^* \mathcal{C}_*[\![PP_*]\!]$.

Solving the Constraints

The constraints generated by $\mathcal{C}_*[\![PP_*]\!]$ can be solved using standard techniques. One possibility is to use a graph formulation of the constraints. The graph will have nodes $\tilde{R}(l)$,

$$\mathcal{C}^\ell_*[\![c^l]\!] = \emptyset$$

$$\mathcal{C}^\ell_*[\![x^l]\!] = \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{\rho}(x) \subseteq \tilde{C}(l)\}$$

$$\mathcal{C}^\ell_*[\![(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0})^l]\!] = \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0})\} \subseteq \tilde{C}(l)\} \cup \mathcal{C}^{l_0}_*[\![t_0^{l_0}]\!]$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0})^l]\!] = \ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0})\} \subseteq \tilde{C}(l)\} \cup \mathcal{C}^{l_0}_*[\![t_0^{l_0}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(l_0) \Rightarrow \{(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0})\} \subseteq \tilde{\rho}(f)\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(t_1^{l_1}\ t_2^{l_2})^l]\!] = \ & \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \mathcal{C}^\ell_*[\![t_2^{l_2}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \{\texttt{on}\} \subseteq \tilde{R}(l_0) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{\rho}(x) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{C}(l_0) \subseteq \tilde{C}(l) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \{\texttt{on}\} \subseteq \tilde{R}(l_0) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{\rho}(x) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{C}(l_0) \subseteq \tilde{C}(l) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{if}\ e\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l]\!] = \ & \mathcal{C}^\ell_*[\![e]\!] \cup \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \mathcal{C}^\ell_*[\![t_2^{l_2}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{C}(l_1) \subseteq \tilde{C}(l)\} \cup \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{C}(l)\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{let}\ x\ \texttt{=}\ t_1^{l_1}\ \texttt{in}\ t_2^{l_2})^l]\!] = \ & \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \mathcal{C}^\ell_*[\![t_2^{l_2}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{C}(l_1) \subseteq \tilde{\rho}(x)\} \cup \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{C}(l)\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{fork}\ t_1^{l_1})^l]\!] = \ & \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \{\texttt{on}\} \subseteq \tilde{R}(l_0) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \{\texttt{on}\} \subseteq \tilde{R}(l_0) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathit{Term}_*\}\end{aligned}$$

$$\mathcal{C}^\ell_*[\![(\texttt{channel}\ t_1^{l_1})^l]\!] = \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{\texttt{channel}\ ()^{l_1}\} \subseteq \tilde{C}(l)\}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{send}\ t_1^{l_1}\ t_2^{l_2})^l]\!] = \ & \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \mathcal{C}^\ell_*[\![t_2^{l_2}]\!] \cup \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{C}(l)\} \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{C}(l_2) \subseteq \tilde{\kappa}(l_0) \mid t = (\texttt{channel}\ ()^{l_0}) \in \mathit{Term}_*\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}^\ell_*[\![(\texttt{receive}\ t_1^{l_1})^l]\!] = \ & \mathcal{C}^\ell_*[\![t_1^{l_1}]\!] \cup \\ & \{\{\texttt{on}\} \subseteq \tilde{R}(\ell) \Rightarrow \{t\} \subseteq \tilde{C}(l_1) \Rightarrow \tilde{\kappa}(l_0) \subseteq \tilde{C}(l) \mid t = (\texttt{channel}\ ()^{l_0}) \in \mathit{Term}_*\}\end{aligned}$$

Table 5: Constraint Generation ($\mathcal{C}^\ell_*$)

$\tilde{C}(l)$, $\tilde{\rho}(x)$ and $\tilde{\kappa}(l)$ for $l \in \mathit{Lab}_*$ and $x \in \mathit{Var}_*$. Associated with each node q we have a data field $\mathsf{D}[q]$ that initially is given by

$$\mathsf{D}[q] = \{u \mid (\{u\} \subseteq q) \in \mathcal{C}_*[\![PP_*]\!]\}$$

The graph will have edges for a subset of the constraints in $\mathcal{C}_*[\![PP_*]\!]$; each edge will be decorated with the constraint that gives rise to it:

- a constraint $q_1 \subseteq q_2$ gives rise to an edge from q_1 to q_2,
- a constraint $\{u\} \subseteq q \Rightarrow q_1 \subseteq q_2$ gives rise to an edge from q_1 to q_2 *and* an edge from q to q_2, and
- a constraint $\{u\} \subseteq q \Rightarrow \{u'\} \subseteq q' \Rightarrow q_1 \subseteq q_2$ gives rise to an edge from q_1 to q_2, an edge from q to q_2 *and* an edge from q' to q_2.

Thus the resulting graph has $O(n)$ nodes and at most $O(n^2)$ edges.

Having constructed the graph we now traverse all edges in order to propagate information from one $\mathsf{D}[q_1]$ to another $\mathsf{D}[q_2]$. We make certain only to traverse an edge from q_1 to q_2 when $\mathsf{D}[q_1]$ is extended with a term not previously there. Furthermore, an edge decorated with a constraint $\{u\} \subseteq q \Rightarrow q_1 \subseteq q_2$ is only traversed if in fact $u \in \mathsf{D}[q]$, and similarly an edge decorated with a constraint $\{u\} \subseteq q \Rightarrow \{u'\} \subseteq q' \Rightarrow q_1 \subseteq q_2$ is only traversed if $u \in \mathsf{D}[q]$ as well as $u' \in \mathsf{D}[q']$. For a node q the data field $\mathsf{D}[q]$ is a value in $\mathcal{P}(\mathit{Term}_*)$ so it can be increased at most $O(n)$ times. Since we make sure only to traverse an edge when a new term can be added this gives an overall bound of $O(n^3)$ for solving the constraints, which is the best result known for 0-CFA analysis. Further details of the algorithm may be found in the Appendix.

7 Conclusion

We have shown how the core methodology of abstract interpretation can be adapted to the specification of control flow analyses. The starting point has been an abstract specification of a 0-CFA analysis for CML that is proved seman-

tically correct, and by gradually massaging it we arrive at a syntax-directed procedure for collecting constraints that subsequently can be solved by standard techniques. The various stages of the process have been proved correct; in several cases it was necessary to perform fairly complicated proofs by coinduction. The reason for this complication is that the initial analysis has to be specified as a greatest fixed point in order to enjoy the Moore family property, whereas the resulting syntax-directed analysis is naturally defined as a least fixed point.

The 0-CFA analysis is indeed a rather simple analysis in that it does not distinguish between the various occurrences of program points and variables and hence it captures no aspects of causality. However, we conjecture that parts of the overall methodology can be applied to more sophisticated control flow analyses [13].

Acknowledgements This work is partly funded by *LOMAPS* (ESPRIT BRA Project) and *DART* (Danish Natural Science Research Council Project). The final version of this paper benefited from discussions with Torben Amtoft.

Appendix

This appendix is somewhat technical and may be skipped on a first reading.

Proof of Theorem 1

Lemma 6 If $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[ie_1]$ then we also have that $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ie_1$. □

Lemma 7 If $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[it_1^{l_1}]$ and $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K it_2^{l_1}$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[it_2^{l_1}]$. □

The proofs are by straightforward induction on the evaluation context.

Lemma 8 If $\rho \vdash it_1^{l_1} \to it_2^{l_2}$ then $l_1 = l_2$. □

The proof is by a straightforward inspection of the inference.

Proposition 9 If $\mathcal{E}(\emptyset, E) \vdash ie_1 \to ie_2$ and $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[ie_1]$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[ie_2]$. □

Proof First we show that if $\rho \vdash ie_1 \to ie_2$, $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ie_1$ and $\rho \ \mathcal{R}_K \ \widehat{\rho}$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K ie_2$ by inspection of the inference $\rho \vdash ie_1 \to ie_2$. Next we establish that if $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K E[ie]$ then $\mathcal{E}(\emptyset, E) \ \mathcal{R}_K \ \widehat{\rho}$ by a straightforward induction on the evaluation context, E. Now the Proposition follows from the above and Lemmas 6, 7 and 8.

Theorem 1 *Semantic Correctness*
If $K, PP \longrightarrow K', PP'$ and $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K PP$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_{K'} PP'$. □

Proof The proof is by case analysis on the semantics: for the case (seq) we use Proposition 9 and for the remaining cases we use Lemmas 6 and 7.

Proof of Theorem 2

Proposition 10 Both of the sets $\{(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \mid (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K e\}$ and $\{(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \sqsubseteq (\widehat{C_*^\top}, \widehat{\rho_*^\top}, \widehat{\kappa_*^\top}) \mid (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K e\}$ are Moore families for all $e \in Exp_*$ and are independent of K; their greatest elements are $(\lambda l.Term, \lambda x.Term, \lambda l.Term)$ and $(\widehat{C_*^\top}, \widehat{\rho_*^\top}, \widehat{\kappa_*^\top})$, respectively; also they have the same least element. □

Proof A straightforward application of coinduction.

Theorem 2 *Moore Family Property*
Both of the sets $\{(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \mid (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K PP_*\}$ and $\{(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \sqsubseteq (\widehat{C_*^\top}, \widehat{\rho_*^\top}, \widehat{\kappa_*^\top}) \mid (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K PP_*\}$ are Moore families and are independent of K; their greatest elements are $(\lambda l.Term, \lambda x.Term, \lambda l.Term)$ and $(\widehat{C_*^\top}, \widehat{\rho_*^\top}, \widehat{\kappa_*^\top})$, respectively; also they have the same least element. □

Proof Immediate by Proposition 10.

Proof of Theorem 3

Proposition 11 For all expressions $e \in Exp_*$ and all labels $\ell \in Lab_*$: if $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* e$ then $(\widehat{R_*^\top}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*\ell} e$; and if $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*\ell} e$ and $\mathtt{on} \in \widehat{R}(\ell)$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* e$. □

Proof For the first result we proceed as follows. First write

$$(\widehat{C}, \widehat{\rho}, \widehat{\kappa}, K, l, e) \in \models_2^* \text{ for } (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* e$$
$$(\widehat{C}, \widehat{\rho}, \widehat{\kappa}, K, l, e) \in \models_3^* \text{ for } (\widehat{R_*^\top}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l} e$$

where the relation $\models_K^*$ is defined in Table 2 and the relation $\models^{*l}$ is defined in Table 3. Then note that these tables define monotonic functions F_2 and F_3 such that $\models_2^* = gfp(F_2)$ and $\models_3^* = gfp(F_3)$. Next we prove

$$F_2(\models_2^*) \subseteq F_3(\models_2^*)$$

by inspection of all clauses. Finally $\models_2^* \subseteq \models_3^*$ follows by coinduction.

For the second result we write

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}, K, l, e) \in \models_2^* \text{ for } (\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* e$$
$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}, K, l, e) \in \models_3^* \text{ for } (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l} e$$
$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}, K, l, e) \in \mathtt{ON} \text{ for } \mathtt{on} \in \widehat{R}(l)$$

where again $\models_2^* = gfp(F_2)$ and $\models_3^* = gfp(F_3)$ for monotonic functions F_2 and F_3 given by Tables 2 and 3. Next we prove

$$F_3(\models_3^*) \cap \mathtt{ON} \subseteq F_2(\models_3^* \cap \mathtt{ON})$$

by inspection of all clauses. This shows $\models_3^* \cap \mathtt{ON} \subseteq F_2(\models_3^* \cap \mathtt{ON})$ and hence $\models_3^* \cap \mathtt{ON} \subseteq \models_2^*$ follows by coinduction.

Theorem 3 *Preservation of Solutions*
If $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* PP_*$ then $(\widehat{R_*^\top}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$. If $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$ then $(\widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_K^* PP_*$. Here PP_* is as above and K is arbitrary. □

Proof Immediate by Proposition 11.

Proof of Theorem 4

Lemma 12 If $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s PP_*$ then (i) $(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ ensures $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{l_0} t_0^{l_0}$, and (ii) $(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ ensures $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{l_0} t_0^{l_0}$ and $\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f)$. □

Proof The result is immediate from the syntax-directed nature of Table 4 and the construction of $Term_*$.

Lemma 13 Suppose that all labels of PP_* are unique. If $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ is *least* such that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$ then (i) $(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ ensures $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_0} t_0^{l_0}$, and (ii) $(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ ensures $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_0} t_0^{l_0}$ and $\texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f)$. □

Proof Sketch Let $\mathcal{L}$ be the set of those $l_0 \in Lab_*$ violating the above. If some $l_0 \in \mathcal{L}$ has $\texttt{on} \in \widehat{R}(l_0)$ we proceed as follows. Let $\widehat{R}' = \widehat{R}[l_0 \mapsto \emptyset]$ and note that by construction of $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ we have $(\widehat{R}', \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \not\models^* PP_*$. Since $\models^*$ is defined coinductively (or to be precise: each $\models^{*\ell}$ is) this means that $\not\models^*$ is defined inductively (or to be precise: each $\not\models^{*\ell}$ is). There must be some $PP_*(p) = t^l$ such that a finitary unfolding of the clauses for $(\widehat{R}', \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l} t^l$ shows that $(\widehat{R}', \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ is not a solution (because $l \neq l_0$ thanks to the uniqueness of labels). Inspection of the defining clauses of $\models^{*\ell}$ in Table 3 show that this must occur in (app) or (fork) because of a condition $\texttt{on} \in \widehat{R}'(l') \Rightarrow \texttt{on} \in \widehat{R}'(l_0)$; in other words $\texttt{on} \in \widehat{R}'(l')$ but $\texttt{on} \notin \widehat{R}'(l_0)$.

In the analogous unfolding of $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l} t^l$ the condition $\texttt{on} \in \widehat{R}(l') \Rightarrow \texttt{on} \in \widehat{R}(l_0)$ no longer is offending and inspection of (app) and (fork) ensures that one of

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_0} t_0^{l_0} \text{ or}$$

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_0} t_0^{l_0} \wedge$$
$$\text{hspace*1cm } \texttt{on} \in \widehat{R}(l_0) \Rightarrow (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \widehat{\rho}(f)$$

is established as required — depending on whether $(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ or $(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$ (using again that labels are unique) — and hence $l_0 \notin \mathcal{L}$. This shows that all $l_0 \in \mathcal{L}$ have $\widehat{R}(l_0) = \emptyset$.

Next define $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models'^{l_0} t_1^{l_1}$ (where $\widehat{R}$, $\widehat{C}$, $\widehat{\rho}$, and $\widehat{\kappa}$ are fixed) to mean

$$\begin{array}{l}((\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_0} t_1^{l_1}) \vee \\ ((\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_* \wedge l_0 \in \mathcal{L} \\ \qquad \wedge\ t_1^{l_1} \text{ is an exposed subexpression of } t_0^{l_0}) \vee \\ ((\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_* \wedge l_0 \in \mathcal{L} \\ \qquad \wedge\ t_1^{l_1} \text{ is an exposed subexpression of } t_0^{l_0})\end{array}$$

where an exposed subexpression is one that is *not* occurring in the body of any $\texttt{fn}\ x' \texttt{=>} \cdots$ or $\texttt{fun}\ f'\ x' \texttt{=>} \cdots$ in $t_0^{l_0}$.

We now want to show that $\models' \subseteq F_3(\models')$ where F_3 is the monotonic function defined by Table 3 such that $\models^* = gfp(F_3)$; coinduction then gives $\models' \subseteq \models^*$ showing that $\mathcal{L}$ must be empty. Since $\models^* \subseteq F_3(\models^*) \subseteq F_3(\models')$ there are only two cases to consider.

The first case is where $(\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in Term_*$, $l_0 \in \mathcal{L}$, and $t_1^{l_1}$ is an exposed subexpression of $t_0^{l_0}$, and where we must establish $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})\ F_3(\models')^{l_0}\ t_1^{l_1}$. We prove this by inspection of the clauses of Table 3. Since $\widehat{R}(l_0) = \emptyset$ all conditions of the form $(\texttt{on} \in \widehat{R}(l_0) \Rightarrow \cdots)$ are immediate; also all $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models'^{l_0} t_2^{l_2}$ that are not inside the scope of a universal quantifier are immediate by the construction of $\models'$; finally we are left with considering the "bodies" of the universal quantifiers appearing in rules (app) and (fork). Here the universal quantifier ranges over some $(\texttt{fn}\ x_2\ \texttt{=>}\ t_2^{l_2}) \in Term_*$ or $(\texttt{fun}\ f_2\ x_2\ \texttt{=>}\ t_2^{l_2}) \in Term_*$ and we must prove $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models'^{l_2} t_2^{l_2}$ and possible also $\texttt{on} \in \widehat{R}(l_2) \Rightarrow (\texttt{fun}\ f_2\ x_2\ \texttt{=>}\ t_2^{l_2}) \in \widehat{\rho}(f_2)$; if $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^{*l_2} t_2^{l_2}$ then the first claim is immediate and otherwise $l_2 \in \mathcal{L}$ in which case the first claim follows from construction of $\models'$; the second claim is immediate unless $\texttt{on} \in \widehat{R}(l_2)$ in which case $l_2 \notin \mathcal{L}$ and hence the desired result follows from construction of $\mathcal{L}$.

The second case is where $(\texttt{fun}\ f_2\ x_2\ \texttt{=>}\ t_2^{l_2}) \in Term_*$ and is similar.

Theorem 4 *Preservation of Solutions*
Suppose that all labels of PP_* are unique. If $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^* PP_*$ then $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$. If $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ is *least* such that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models^* PP_*$ then $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^* PP_*$. □

Proof We write $\models_3^* = gfp(F_3)$ for $\models^*$ and $\models_4^* = gfp(F_4)$ for $\models_s^*$. For the first result we prove

$$F_4(\models_4^*) \subseteq F_3(\models_4^*) \tag{4}$$

as coinduction then gives $\models_4^* \subseteq \models_3^*$ as desired. To prove (4) we simply inspect the clauses; most clauses are straightforward except for (app) and (fork) where we make use of Lemma 12.

For the second result we prove

$$F_3(\models_3^*) \subseteq F_4(\models_3^*) \tag{5}$$

as coinduction then gives $\models_3^* \subseteq \models_4^*$ as desired. To prove (5) we simply inspect the clauses; most clauses are straightforward except for (fn) and (fun) where we make use of Lemma 13.

Proof of Theorem 5

Proposition 14 $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{*\ell} e$ if and only if $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c^* \mathcal{C}_*^\ell[\![e]\!]$. □

Proof A simple structural induction on e shows that

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^{*\ell} e \text{ if and only if } (\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c^* \mathcal{C}_*^\ell[\![e]\!]$$

for all $e \in Exp_*$.

Theorem 5 *Preservation of Solutions*
$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_s^* PP_*$ if and only if $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c^* \mathcal{C}_*[\![PP_*]\!]$. □

Proof Immediate by Proposition 14.

INPUT: $\mathcal{C}_*[\![PP_*]\!]$

OUTPUT: $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$

METHOD: Step 1: Initialisation

```
W := nil;
for q in Nodes do D[q] := ∅;
for q in Nodes do E[q] := nil;
```

Step 2: Building the graph

```
for c in C*[[PP*]] do
    case c of
        {u} ⊆ q: add(q,{u});
        q1 ⊆ q2: E[q1] := cons(c,E[q1]);
        {u1} ⊆ q1' ⇒ ··· ⇒ {uk} ⊆ qk' ⇒ q1 ⊆ q2:
            E[q1] := cons(c,E[q1]); E[q1'] := cons(c,E[q1']); ··· E[qk'] := cons(c,E[qk']);
```

Step 3: Iteration

```
while W ≠ nil do
    q := head(W); W := tail(W);
    for c in E[q] do
        case c of
            q1 ⊆ q2: add(q2, D[q1]);
            {u1} ⊆ q1' ⇒ ··· ⇒ {uk} ⊆ qk' ⇒ q1 ⊆ q2:
                if u1 ∈ D[q1'] ∧ ··· ∧ uk ∈ D[qk'] then add(q2, D[q1]);
```

Step 4: Recording the solution

```
for l in Lab* do R̂(l) := D[R̃(l)];
for l in Lab* do Ĉ(l) := D[C̃(l)];
for x in Var* do ρ̂(x) := D[ρ̃(x)];
for l in Lab* do κ̂(l) := D[κ̃(l)];
```

USING: `procedure add(q,d) is`

```
if ¬ (d ⊆ D[q])
then  D[q] := D[q] ∪ d;
      W := cons(q,W);
```

Table 6: Algorithm for Solving Constraints.

Constraint Solving

For completeness we present an abstract algorithm for solving constraints. It operates on the main *data structures*:

- a worklist W that is a list of nodes whose outgoing edges should be traversed;
- an array D that for each node gives an element of $\widehat{Val_*}$; and
- an array E that for ech node gives a list of the successor nodes.

The set Nodes contains $\tilde{R}(l)$, $\tilde{C}(l)$, and $\tilde{\kappa}(l)$ for all l in Lab_* and $\tilde{\rho}(x)$ for all x in Var_*.

The first step of the algorithm is to suitable initialise the data structures. The second step is to build the graph and to perform the initial assignments to the data fields. This is established using the procedure add(p,d) that incorporates d into D[p] and adds p to the work list if d was not part of D[p]. The third step is to continue propagating contributions along edges as long as the work list is non-empty. The fourth and final step is to record the solution in a more familiar form.

The following result shows that the algorithm does indeed compute the solution we want:

Proposition 15
Given input $\mathcal{C}_*[\![PP_*]\!]$ the algorithm of Table 6 terminates and the result $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa})$ produced by the algorithm satisfies

$$(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \quad = \quad \sqcap\{(\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \mid (\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \models_c \mathcal{C}_*[\![PP_*]\!]\}$$

and hence is the least solution to $\mathcal{C}_*[\![PP_*]\!]$. □

Proof It is immediate that Steps 1, 2 and 4 terminate, and this leaves us with Step 3. It is immediate that the values of D[q] never decrease and that they can be increased at most a finite number of times. It is also immediate that a node is added to the work list only if some value of D[q] actually increased. To each node placed on the work list only a finite amount of calculation (bounded by the number of outgoing edges) needs to be performed in order to remove the node from the work list. This guarantees termination.

Next let $(\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1)$ be a solution to $(\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \models_s \mathcal{C}_*[\![PP_*]\!]$. It is possible to show that the following invariant

$$\forall l \in Lab_* : \mathsf{D}[\tilde{R}(l)] \subseteq \widehat{R_1}(l)$$
$$\forall l \in Lab_* : \mathsf{D}[\tilde{C}(l)] \subseteq \widehat{C_1}(l)$$
$$\forall x \in Var_* : \mathsf{D}[\tilde{\rho}(x)] \subseteq \widehat{\rho}_1(x)$$
$$\forall l \in Lab_* : \mathsf{D}[\tilde{\kappa}(l)] \subseteq \widehat{\kappa}_1(l)$$

is maintained at all points after Step 1. It follows that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \sqsubseteq (\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1)$ upon completion of the algorithm.

We prove that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c \mathcal{C}_*[\![PP_*]\!]$ by contradiction. So suppose there exists $c \in \mathcal{C}_*[\![PP_*]\!]$ such that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c c$ does *not* hold. If c is $\{u\} \subseteq q$ then Step 2 ensures that $\{u\} \subseteq \mathsf{D}[q]$ and this is maintained throughout the algorithm; hence c cannot have this form. If c is $q_1 \subseteq q_2$ it must be the case that the final value of D satisfies $\mathsf{D}[q_1] \neq \emptyset$ since otherwise $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c c$ would hold; now consider the last time $\mathsf{D}[q_1]$ was modified and note that q_1 was placed on the work list at that time (by the procedure add); since the final work list is empty we must have considered the constraint c (which is in $\mathsf{E}[q_1]$) and updated $\mathsf{D}[q_2]$ accordingly; hence c cannot have this form. If c is $\{u_1\} \subseteq q_1' \Rightarrow \cdots \{u_k\} \subseteq q_k' \Rightarrow q_1 \subseteq q_2$ it must be the case that the final value of D satisfies $\mathsf{D}[q_1']$ $\neq \emptyset$, $\cdots$, $\mathsf{D}[q_k'] \neq \emptyset$ and $\mathsf{D}[q_1] \neq \emptyset$; now consider the last time one of $\mathsf{D}[q_1']$, $\cdots$, $\mathsf{D}[q_k']$ and $\mathsf{D}[q_1]$ was modified and note that q_1', $\cdots$, q_k', or q_1 was placed on the work list at that time; since the final work list is empty we must have considered the constraint c and updated $\mathsf{D}[q_2]$ accordingly; hence c cannot have this form.

We have now shown that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \models_c \mathcal{C}_*[\![PP_*]\!]$ and that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) \sqsubseteq (\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1)$ whenever $(\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \models_c \mathcal{C}_*[\![PP_*]\!]$. It now follows that $(\widehat{R}, \widehat{C}, \widehat{\rho}, \widehat{\kappa}) = \sqcap\{(\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \mid (\widehat{R}_1, \widehat{C}_1, \widehat{\rho}_1, \widehat{\kappa}_1) \models_c \mathcal{C}_*[\![PP_*]\!]\}$ as required.

References

[1] J.M. Ashley. A practical and flexible flow analysis for higher-order languages. In *Proc. 23th POPL*, pages 184–195. ACM Press, 1996.

[2] P. Cousot and R. Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, pages 238–252. ACM Press, 1977.

[3] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th POPL*, pages 269–282. ACM Press, 1979.

[4] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. FPCA '95*, pages 170–181. ACM Press, 1995.

[5] K.-F. Faxén. Optimizing lazy functional programs using flow inference. In *Proc. SAS '95*, pages 136–153. SLNCS 983, 1995.

[6] C. Flanagan and M. Felleisen. The semantics of Future and its use in program optimization. In *Proc. POPL*. ACM Press, 1995.

[7] N. Heintze. Set-based analysis of ML programs. In *Proc. Lisp and Functional Programming*. ACM Press, 1994.

[8] S. Jagannathan and S. Weeks. Analysing stores and references in a parallel symbolic language. In *Proc. Lisp and Functional Programming*. ACM Press, 1994.

[9] S. Jagannathan and S. Weeks. A unified treatment of flow analysis in higher-order languages. In *Proc. POPL '95*. ACM Press, 1995.

[10] S. Jagannathan and A. Wright. Effective flow analysis for avoiding run-time checks. In *Proc. SAS '95*, pages 207–224. SLNCS 983, 1995.

[11] N.D. Jones and F. Nielson. Abstract Interpretation: a semantics-based tool for program analysis. In *Handbook of Logic in Computer Science* **vol. 4**. Oxford University Press, 1995.

[12] R. Milner, M. Tofte and R. Harper. The Definition of Standard ML. The MIT Press, Cambridge, Mass, 1990.

[13] F. Nielson and H.R. Nielson. Infinitary Control Flow Analysis: a Collecting Semantics for Closure Analysis. *Proc. POPL*, pages 332–345, 1997.

[14] H.R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology. In *Proc. POPL '94*, pages 84–97. ACM Press, 1994.

[15] J. Palsberg. Closure analysis in constraint form. *ACM TOPLAS*, 17 (1):47–62, 1995.

[16] G.D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Aarhus University, Denmark, 1981.

[17] J.H. Reppy. Concurrent ML: Design, application and semantics. In *Proc. Functional programming, Concurrency, Simulation and Automated Reasoning*, pages 165–198. SLNCS 693, 1993.

[18] P. Sestoft. *Analysis and efficient implementation of functional programs*. Ph.D.-thesis, Department of Computer Science, University of Copenhagen, Denmark, 1991.

[19] O. Shivers. Control flow analysis in Scheme. In *Proc. PLDI'88*, pages 164–174. ACM Sigplan Notices 7 (1), 1988.

[20] O. Shivers. The semantics of Scheme control-flow analysis. In *Partial Evaluation and Semantics-Based Program Manipulation*. ACM SIGPLAN Notices 26 (9), 1991.

[21] M. Wand and P. Steckler. Selective and lightweight closure conversion. In *Proc. POPL '94*, pages 435–445. ACM Press, 1994.