Flow Logics for Mobile Ambients

René Rydhof Hansen Jacob Grydholt Jensen

May 1999

In this thesis it is studied how static analysis in general, and flow logics in particular, can be applied to the ambient calculus. A flow logic for the ambient calculus is presented and used to guarantee security properties for systems modelled in the ambient calculus. Furthermore it is shown how to generate constraints equivalent to the specification of the analysis. This leads to the presentation of a worklist algorithm that solves the constraints in low polynomial time. A web-based prototype implementing the worklist algorithm is also presented; the prototype automatically generates and solves the constraints for a given process in the ambient calculus and presents the result of the analysis. This result can furthermore be used to automatically check whether the process is a firewall.

Contents

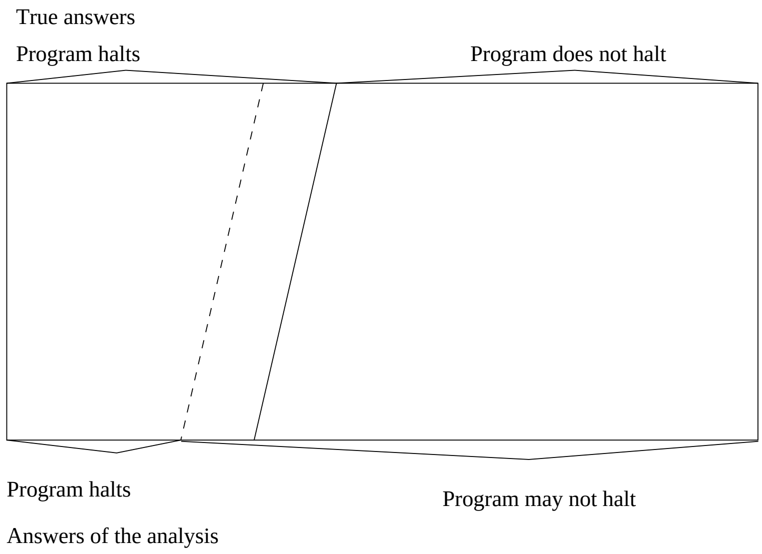
Chapter 1

Introduction

Program analysis is a static technique to obtain information about aspects of the dynamical behaviour of programs. This information can be used in for example optimising compilers or to validate systems. As it is a static approach it is very often restricted to giving approximate answers. For instance one aspect of the dynamical behaviour of programs is whether a program will halt on a given input or not. Clearly if the language at hand is Turing complete, no static analysis will ever be able to give the correct answer for all programs. The aim of a program analysis is to provide safe answers which are as precise as possible. Let us consider the case of an analysis which gives one of the following answers to a program.

- The program may not halt
- The program halts

The aspect of safety is illustrated by the following figure:



We see that the programs for which the analysis answers “Halts” do in fact halt.

Approaches to program analysis. Traditionally there are a number of approaches to program analysis: data flow analysis, control flow analysis, abstract interpretation and type and effect systems (see [NNH] for a textbook treatment of these areas). Data flow analysis was the earliest and has traditionally been used in optimising compilers [ASU86] for imperative languages where the flow of control is easily calculated.

For functional and object-oriented languages the flow of control is less obvious due to dynamic dispatch. In functional languages the use of higher-order functions means that the set of functions that can be applied at an application point is non-trivial to determine. This problem is also found in object-oriented languages containing virtual methods (for instance Java). Program analysis must here find a way to decide which function/methods can be called or invoked. When the flow of control is not easily decidable from the syntax of a program, it is necessary to perform a control flow analysis which can be combined with a data flow analysis.

Traditionally, analyses are specified and then proved correct. Abstract interpretation approaches the issue of correctness in another way. In abstract interpretation one typically specifies a collecting semantics which captures the information that one is seeking to obtain. The collecting semantics is undecidable in most cases, cf. the discussion on Turing completeness above, but it is relatively easy to prove its correctness. From the collecting semantics, program analyses can be calculated using mathematical machinery which ensures that the correctness of the collecting semantics carries over. In this way one starts with an undecidable collecting semantics which is proven correct, and ends with a correct analysis which can be effectively implemented.

Type and effect systems have also been used to analyse programs. In [CG99] Mobile Ambients is given a type system and this information is used to predict behaviour of processes. They have also been used for so-called region inference which can be used to implement functional languages in a stack-based regime.

Applications. A number of applications for the result of program analysis exists. The most traditional application is in program optimisation. For instance when one analyses a Java program and the analysis states that for a method invocation only one method can be called, the compiler might perform a direct call to this method instead of a costly table-lookup. We have also brushed upon the use of region inference to implement functional languages. As an example the ML-kit [TBE+98] uses region inference in order to implement a Standard ML compiler.

Recently, program analysis has also been used within the field of software validation. When a company hires subcontractors to produce part of a software system, they could for instance want some control over that the produced part does not violate for instance security.

1.1 Program analysis and Mobile Ambients

In order to study the notions, such as mobile computing and mobile computation, that have emerged with the increased use of large scale networking, e.g. the Internet and WWW, Cardelli and Gordon have developed a calculus of Mobile Ambients. The calculus of Mobile Ambients can be seen as a process algebra that provides named locations (ambients) for processes, in contrast to other well known process calculi such as CCS and the π-calculus. These locations can be arbitrarily nested, and the nesting can be changed dynamically, making the Mobile Ambients calculus very well suited as a formalism for studying mobility.

It should be evident that this arbitrary nesting easily becomes very complex and it becomes very hard, if not impossible, to manually keep track of how these ambients are nested, and what hierarchies they are organised in. In this thesis we propose a control flow analysis of the ambients calculus as a way to overcome these problems.

The above mentioned increase in the use of the Internet and WWW has brought with it an increased awareness of the importance of security issues. These issues are of particular importance in systems that allows access for mobile agents. One possible solution to (some of) the security problems, is to formulate formal security policies and *formally* verify that these hold in the systems of interest. Since formally verifying that certain properties hold for a given system can be a very time consuming task, automating such tasks, or parts of them, would be very helpful. In this thesis we define a number of formal security properties for systems modelled in the mobile ambients calculus, and it is shown how the analysis may be used to construct static checks for these properties.

The Mobile Ambients calculus as presented in [CG97a, CG98] extends the basic calculus with primitives for simple communication. In this thesis we shall focus our attention on the (Turing complete) basic calculus.

1.2 Overview of the thesis

In Chapter 2 we give a brief presentation of different approaches to static analysis including flow logics.

Chapter 3 introduces the syntax and semantics of the calculus to be studied: the Mobile Ambients calculus.

In Chapter 4 we present a syntax-directed specification of what an acceptable analysis is of a process in the Mobile Ambients calculus without communication. An acceptable analysis provides information about the movement of ambients and capabilities inside the process. The specification is within the framework of flow logics. This framework was chosen as it provides for concise, easily understandable specifications and has been successfully applied to other calculi [BDNN98, NN98b, NN97].

The correctness of the specification is proved by a subject reduction result; this approach is well-known from type systems. It is also proved that all processes not only admits an analysis but that a best analysis also exists. This is done by a so-called Moore-family result.

In Chapter 5 we study how the development from the previous chapters may be applied to the field of computer security.

Chapter 6 presents a technique for finding the best analysis to a given process. This involves producing set constraints which are solved by an independent solver. We show that straight-forward production of constraints leads to a $O(s^5)$ algorithm (assuming that the size of the process is s) which is improved to yield a $O(s^4)$ algorithm which is comparable to control flow analyses of other calculi.

In order to get a feel of the analysis and its applications a prototype with a web interface has been developed. The prototype, described in Chapter 7, implements the algorithm of Chapter 6 in order to compute the least analysis. The prototype includes the possibility to check properties introduced in Chapter 5. It should also be noted that a large number of examples in the thesis have been produced from the prototype.

Chapter 2

Static Analysis

As was noted in the introduction, static analysis is a way to statically determine approximations of dynamic behaviour. Several techniques exist for doing this: Abstract interpretation, inference based, constraint based, data flow analysis, grammar flow and abstract reduction, to mention only a few. Traditionally static analyses have been used primarily in optimising compilers to validate program transformations and while this may still be the primary use of static analysis techniques, they have also found use in other areas, such as software validation, program understanding and, more recently, systems security. As an example of using (inference based) static analysis in the area of program understanding, in [EHM+99] the Year 2000 problem is formulated as a static analysis problem.

In this thesis, we investigate how static analysis can be used to guarantee certain security aspects modelled in a calculus intended for modelling mobility aspects of computation: The ambient calculus.

2.1 Data flow analysis

Data flow analysis is the best known of the approaches to program analysis that we consider. Data flow analysis is described in many textbooks on compiler writing, e.g. [ASU86]. Text book treatments of data flow analysis techniques can be found in [NNH, Hec77].

Data flow analysis is the process of ascertaining and collecting information about the modification, preservation, and use of "quantities" in a computer program. Typically, (values of) variables are selected as the quantities under study, because they provide very fundamental information from which other information can be inferred.

In data flow analysis it is customary to think of a program as a graph. The potential flow of control of each procedure is therefore typically represented by a directed graph called a *control flow graph*, which depicts all possible execution paths. The nodes of such a graph may represent individual statements or sequences of statements.

Classically, data flow analyses are divided into *forwards* and *backwards* analyses. For forwards analyses information is propagated in the same direction as control flow, whereas in backwards analyses information is propagated in the opposite direction of control flow. Furthermore, analyses are divided into *may* and *must* analyses. For a may analysis we are able to detect properties satisfied by *at least one* path to each node. For a must analysis we are able to detect properties satisfied by *all* paths to each node.

In data flow analysis one often takes an *equational* approach in which the analysis is specified by extracting a number of equations from a program. There are two classes of equations. In forwards analyses one class of equations relates exit information of a node in the control flow graph to entry information for the same node. The other class of equations relate entry information of a node to exit information of nodes from which there is an edge to the node of interest, i.e. entry information is obtained from all the exit information where control could have come from. The result of the analysis is then obtained by choosing an appropriate solution to the equations. For backwards analyses the same approach is used where the roles of entry and exit nodes are reversed.

An example of a forwards, must analysis is the *Available Expressions Analysis*. It determines for each node in the control flow graph, which expressions must have already been computed, and not later modified on all paths to the entry of the node. This information can be used to avoid the re-computation of an expression.

An example of a backwards, may analysis is the *Live Variables Analysis*. This analysis determines which variables are *live* at the exit from a node. A variable is live if there exists a path from the node to a use of the variable that does not re-define the variable. This information can be used to eliminate so-called dead code.

For completeness we mention the two other classical data flow analyses, *Reaching Definitions*, which is a forwards, may analysis and *Very Busy Expressions*, which is a backwards, must analysis.

We have tried to illustrate that most data flow analyses share a common base. This has lead to the *monotone framework* introduced in [KU77]. For an analysis in the monotone framework, one specifies the data flow values, the direction of the analysis (forwards or backwards), how information for several nodes are combined (corresponds to the difference between may and must analyses above) and finally transfer functions which specify how information is propagated through the nodes (corresponds to the equations above that relate information inside nodes).

The advantages that arise from identifying such a framework include the possibility of creating analysis generators with generic algorithms for solving the data flow equations. As an example the PAG system [Mar98] automatically generate an implementation of an analysis specified in the monotone framework.

So far we have only discussed intraprocedural analysis, which concerns analysis within a single procedure; in the following we will discuss interprocedural analysis, which takes procedure calls into consideration.

First we consider some simple approaches to interprocedural analysis. One such method is termed the *worst case method*. According to this method we make "worst case" assumptions about the effect of each procedure calls; this might lead to invalidating all available information at each procedure call making the analysis grossly imprecise. Another approach is the *inlining method*. For this method every call to a procedure is replaced by the body of that procedure. The resulting program is then analysed using intraprocedural techniques. This is only feasible for non-recursive procedures and the control flow graph grows exponentially in terms of the nesting depth of the procedure calls.

An approach similar to intraprocedural analysis represents the calling relationships among the procedures of a program by a directed graph named a *call graph*, which shows "what may directly call what" one or more times. Interprocedural analysis on a call graph is much harder than intraprocedural analysis on a control flow graph since the information that flows into a called procedure from different call points preferably should flow back to the appropriate return points.

2.2 Abstract Interpretation

Abstract interpretation is a very flexible and powerful framework for constructing program analyses; as a consequence of this, and unlike data flow analysis, it is very hard to give any general categorisation of abstract interpretations. In what follows we shall first describe the basic idea of abstract interpretation. See [NNH, WM95] for textbook treatments.

Abstract semantics. The essential idea in abstract interpretation is that of an abstract or non-standard semantics, based on a non-standard interpretation of constants and operators in the language analysed. This non-standard semantics is intended to capture the properties and notions of interest for the analysis. The abstract semantics should of course be sound with respect to the standard semantics; this is often established through a correctness relation that relates the concrete data or operator in the standard domain to some abstract data or operator in the abstract domain. These domains are typically complete lattices. Alternatively, the soundness can be expressed by using representation functions. Intuitively, a representation function maps a concrete value to the abstract value that *best* describes the concrete value. These two formulations can be proved equivalent, cf. [NNH]. Often the the abstract semantics can be specified as a least fixed point of a suitable continuous operator on a suitable domain of (abstract) values, cf. [CC92].

It may well happen that the calculation of the least fixed point is prohibitively expensive, rendering the analysis useless on all but the smallest programs. In abstract interpretation there exist two techniques for overcoming this problem by systematically constructing less costly analyses from a given analysis, this comes at the price of a less precise analysis. An important point worth noting about the techniques is that both preserve semantic soundness of the analysis; thus once an analysis has been proven sound with respect to the standard semantics, any analyses constructed from the original analysis are, by default, also semantically sound with respect to the standard semantics.

Galois connections. The first of these methods basically constructs new and less complex abstract domains from the original abstract domain; based on the assumption that the calculations necessary to perform an analysis are cheaper in the simpler domains. Technically the simpler domains are constructed through the concept of a Galois connection which ensures that semantic soundness of the analysis is preserved. In [NNH] a wide variety of ways to construct Galois connections are described.

Widenings and narrowings. In some cases the least fixed point, representing the abstract semantics, can be computed by iteration (cf. [CC92]) of the operator from a suitable basis, in other cases the iteration can not be guaranteed to stabilise. In order to achieve stabilisation in the latter case and speed up computation in the former, the widening operator is used to transform the iteration sequence into a sequence that is guaranteed to stabilise at a sound approximation of the least fixed point; due to the approximative nature of the transformation this also speeds up the computation in in the case where the original sequence would also have stabilised, at the price of less precision. Thus widenings are targeted at the computation of the least fixed point itself rather than its domain.

The imprecision caused by the use of the widening operator can be alleviated by the use of a narrowing operator. Where a widening operator can be seen as a sort of upward approximation, a narrowing operator is a sort of downward approximation. Thus once a sequence has stabilised at some sound approximation of the least fixed point, typically through the use of a widening operator, the narrowing operator is used to improve the precision of the result.

Collecting semantics. The concepts discussed above can be used to construct abstract interpretations in a very methodical way. Rather than constructing an ad-hoc abstract semantics, a so-called *collecting semantics* is constructed. The collecting semantics is intended as a sort of trace semantics tracking the properties of interest for the analysis. Often the collecting semantics is undecidable, but often designed to be almost trivially sound with respect to the standard semantics. This collecting semantics is used as a basis for constructing less precise, and hopefully decidable, analyses using the techniques described above. With this methodology an entire design space of analyses is created, with the collecting semantics at the root.

In [HJNN99] we have used this methodology, expressed in the terminology of flow logics, to create a number of analyses for the ambient calculus (see Chapter 3) starting with a collecting semantics.

2.3 Inference based

In the inference based approach to static analysis (annotated) type systems and effect systems are used to infer not only type information, but also analysis information. This approach has been used for a wide variety of analyses: in [Aba97] and [HR98] it is used to guarantee certain security properties in the spi-calculus and the SLam calculus respectively; in [EHM$^+$99] a special type system is used to identify year 2000 problems in Cobol programs.

As was hinted at above, the basic idea in inference based analyses is rather simple: annotate the type system with program analysis information. This often leads to a very clean specification of the desired program analysis. In addition, there is a clear and natural distinction between the specification of acceptable analyses and the task of actually finding an analysis for a particular program. The distinction reflects a similar distinction in (ordinary) type systems, namely between type checking and type reconstruction.

When analysing higher order languages with side effects, such as Standard ML, it is often necessary to annotate the "arrow", i.e. the function type, with an *effect*, representing the possible side effects that could occur when applying a particular function; this is the case in [HR98] where an analysis of a λ-calculus with reference cells and concurrency primitives, called the SLam calculus, is constructed and is used to guarantee that no security violations can occur.

Standard type-theoretic notions such as polymorphism and subtyping are also useful when using type systems for program analysis, and can be used to increase the precision of an analysis.

One potential drawback of type and effect systems is that is may be very hard to implement an analysis, i.e. to implement type reconstruction. For effect systems the decidability of the analysis/type systems depends (also) on the structure of the effects used for annotation, since it is necessary to reconstruct not only type information but also analysis information and in particular effect information. Another potential drawback of this approach, also related to implementing an analysis, is that often it is not obvious whether or not principal types exist for a given type system, cf. [Aba97, HR98].

A final example of the use of type systems of particular relevance is [CGG99] where a type system is used to ensure mobility and primitive security properties for the calculus studied in this thesis: the ambient calculus.

2.4 Flow Logic

Flow logic is an approach to static analysis that enables the integration of state-of-the-art insights from data flow analysis and abstract interpretation. Furthermore, in the flow logic approach a clear distinction is made between the specification of the analysis, and the computation of analysis results; this is similar to the type system approach.

Flow logics can often be classified with respect to the following criteria:

- Whether the specification is syntax-directed (compositional) or abstract (non-compositional). The former allows for an inductive definition whereas the latter requires co-inductive methods.
- Whether it is verbose or succinct. The former records all the internal flow information whereas the latter records only top-level information, similar to type systems.

The flow logic approach has been used for a variety of languages and calculi, such as the λ-calculus, cf. [NN97], the imperative objects calculus, cf. [NN98b] and the π-calculus, cf. [BDNN99, BDNN98].

Chapter 3

Introduction to Mobile Ambients

In this chapter we will give a brief introduction to the language under study: the Mobile Ambients calculus. We first motivate the need for a calculus designed specifically with networking and mobility in mind. Next we give a short description of the syntax, slightly modified to facilitate an analysis of the calculus that is described in the following chapters; and finally the semantics of the calculus is introduced and discussed.

3.1 Motivation

With the advent, and ever increasing popularity, of large scale computer networking, e.g. the Internet and the World Wide Web, notions such as *mobile computing* and *mobile computation*, collectively referred to as *mobility*, have emerged. In this context, mobile computing refers to "ordinary" computing carried out on a mobile computer, e.g. notebooks, electronic calendars, handheld computers; while mobile computation refers to programs that are *themselves* mobile and move from location to location while carrying out computations, e.g. applets, agents. A point worth noting about mobility is that mobility as such is not the problem, rather it is in handling privileges and authorisation issues (called *administrative domains* in [CG98, CG97a]): It is not enough merely to know the address of another computer in order to gain access to it; it may well be necessary to supply a password or to pay for it. Furthermore, these administrative domains are typically arranged hierarchically, with each domain or level subject to its own set of restrictions and authorisation procedures. Thus for an agent to move from one point of a network to another, it will typically have to obtain permission in order to leave its own administrative domain(s), and moreover it must obtain permission to enter (and leave) any administrative domain(s) between its starting point and its final destination. This example is equally applicable to the situation where it is not an agent that needs to move through a series of networks, but rather a portable computer that is to be disconnected from its "home" network and reconnected at some "foreign" network that may also involve several administrative domains. It should be obvious that such hierarchies of administrative domains and agents moving through them easily becomes non-trivial, indeed they may well become very complex structures. In order to study and ultimately understand such structures, the first step is to develop formalisms that are suitable for modelling these phenomena. One such formalism, designed specifically with mobility in mind, is Cardelli and Gordon's "Mobile Ambients" cf. [CG98, CG97a].

From the above discussion one can distill a few key notions in mobility: administrative domains, i.e. places or locations with clearly defined boundaries; the ability for agents and applets to move (if allowed), i.e. mobility; and finally ways to allow or reject moves, i.e. authorisation. With this list of key notions in mind, the particular relevance of the ambient calculus should be evident: it was designed to model exactly those concepts, cf. [CG98, CG97a]. This is expressed through the fact that although the calculus basically is a process algebra, it has one particular feature that sets it apart from other process algebras, such as the π-calculus and CCS, namely ambients. As compared to the process algebras mentioned, ambients introduce clear boundaries into the calculus: a clear distinction between "inside" an ambient and "outside". This makes ambients obvious candidates for modelling not only concepts such as locations and places, but also the administrative domains discussed previously. Furthermore, the mobility primitives defined in the calculus are of a nature that lends itself to modelling the remaining key notions: movement is directly encoded as primitives in the calculus, called *capabilities*. There are several kinds of capabilities, corresponding to various ways of navigating through ambients, i.e. the owner of a given movement capability can initiate movement, but only of the kind allowed by the capability. This approach to movement also has the beneficial side effect that agents can only make moves corresponding to their capabilities, thus a capability can be seen as an *authorisation* to make a given move. In Chapter 5 we shall take this idea of authorisation further and study several security related notions and how they apply to processes modelled in the ambient calculus. In that chapter it is also shown how the control flow analysis developed in Chapter 4 can be used to ensure that a given process has certain (security) properties, and indeed how to automatically check for several such properties. In the following we define the syntax and semantics of the ambient calculus.

3.2 Syntax

The point of the flow analysis, specified and discussed in detail in Chapter 4, will be to compute for every ambient the set of ambients and threads that can reach or enter it; and for this purpose we label all such expressions with *ambient labels* (ranged over by $\ell^a, \ell^a_m, \ell^a_n, \ldots$) and *thread labels* (ranged over by $\ell^t, \ell^t_1, \ell^t_2, \ldots$) respectively. Thus let Lab^a denote the set of ambient labels, and Lab^t the set of thread labels. A short (and more technical) explanation of these points is given immediately following the definition of the syntax. For a thorough explanation see Chapter 4. We assume, for convenience, that $\mathsf{Lab}^a \cap \mathsf{Lab}^t = \emptyset$ and define $\mathsf{Lab} = \mathsf{Lab}^a \cup \mathsf{Lab}^t$. For similar reasons it is very useful to also label the *restriction operator*, which is denoted by a ν in the syntax. In order to avoid confusion, these labels are called *markers* and are typically written as: μ, μ_1, μ_2 etc. Let Mar be the set of all markers.

Now let the set of all *ambient names* be denoted by Nam; let $\ell^a \in \mathsf{Lab}^a$, $\ell^t \in \mathsf{Lab}^t$, $\mu \in \mathsf{Mar}$, $n \in \mathsf{Nam}$ and let $P, Q \in \mathsf{Proc}$, where Proc denotes the syntactic category of *processes*, defined as follows:

$$
\begin{array}{lcll}
P, Q & ::= & (\nu_\mu n)P & \text{restriction} \\
 & | & \mathbf{0} & \text{inactivity} \\
 & | & P \mid Q & \text{composition} \\
 & | & !P & \text{replication} \\
 & | & n^{\ell^a}[P] & \text{ambient} \\
 & | & M^{\ell^t}.P & \text{action}
\end{array}
\qquad
\begin{array}{lcll}
M^{\ell^t} & ::= & in^{\ell^t} n & \text{enter } n \\
 & | & out^{\ell^t} n & \text{exit } n \\
 & | & open^{\ell^t} n & \text{open } n
\end{array}
$$

The above definition of the syntax differs from the one given in [CG97a, Section 2] only in that we have added labels to the ambients and capabilities (ℓ^a and ℓ^t respectively), and a marker, μ, to the restriction operator.

As already mentioned above, the reason for these additions to the syntax is in order to facilitate the analysis. Specifically, since the bound names in a term can change through α-conversion (see Definition 3.4) during evaluation, cf. Section 3.3, they are rendered useless for analysis purposes; partly because in order to establish semantic soundness through a subject reduction result, it is a general principle that the analysis information should remain invariant under evaluation. Furthermore, the chosen formulation of the analysis, verbose rather than succinct, necessitates explicit program points, see [NN98a] for a full discussion of these points. To this end labels and markers, inherently invariant under α-conversion and evaluation, are added.

In order to make the syntax of the calculus more legible, we adopt the following notational conventions:

$$
\begin{array}{lll}
(\nu_\mu n)P \mid Q & \text{is read} & ((\nu_\mu n)P) \mid Q \\
!P \mid Q & \text{is read} & (!P) \mid Q \\
M^{\ell^t}.P \mid Q & \text{is read} & (M^{\ell^t}.P) \mid Q
\end{array}
$$

And finally a notational shortcut:

$$M^{\ell^t} \triangleq M^{\ell^t}.\mathbf{0}$$

Thus the term $(\nu_\mu n)P \mid !Q \mid M^{\ell^t}$ is actually read as $((\nu_\mu n)P) \mid (!Q) \mid (M^{\ell^t}.\mathbf{0})$ With these notational conveniences in place, we now give a few examples.

Example 3.1 The following is an example of a term in the syntax. It can be seen as a model of an agent, named n, entering another agent, named m; once inside, agent n is *opened* and the processes/computations inside it is "absorbed" by agent m. Furthermore, the name of agent m is private to m and n (indicated by the ν-operator).

$$(\nu_\mu m)(n^{\ell^a_n}[in^{\ell^t_1} m] \mid m^{\ell^a_m}[open^{\ell^t_2} n])$$

where $\{\ell^a_n, \ell^a_m\} \subseteq \mathsf{Lab}^a$, $\{\ell^t_1, \ell^t_2\} \subseteq \mathsf{Lab}^t$ and $\mu \in \mathsf{Mar}$.

The informal and intuitive meaning of the example term, as given above, is formalised in Section 3.3 and Example 3.8 in particular. $\diamond$

Example 3.2 The example is intended to model an agent accessing a firewall (as defined in [CG98]). First we define two terms modelling the firewall and an agent respectively:

$$\begin{array}{lcl} \mathit{Firewall} & \triangleq & (\nu_\mu w) w^{\ell^a_w}[k_1^{\ell^a_{k_1}}[out^{\ell^t_1} w.in^{\ell^t_2} k_2.in^{\ell^t_3} w] \mid open^{\ell^t_4} k_2.open^{\ell^t_5} k_3.P] \\ \mathit{Agent} & \triangleq & k_2^{\ell^a_{k_2}}[open^{\ell^t_6} k_1.k_3^{\ell^a_{k_3}}[Q]] \end{array}$$

for $P, Q \in \mathsf{Proc}$ such that $(fn(P) \cup fn(Q)) \cap \{k_1, k_2, k_3\} = \emptyset$; where $fn(P) \cup fn(Q)$ is the set of *free names* in P and Q, see Definition 3.3. Thus a system where an agent is trying to gain access to the firewall can be modelled in the following way

$$\mathit{Agent} \mid \mathit{Firewall}$$

In Section 3.3 these informal notions are made precise through a formal definition of the semantics.

This example is [CG98, Section 2.5] rewritten to fit the syntax of this section. $\diamond$

Having defined the syntax of the calculus, we now turn to the definition of several syntax-related notions needed in subsequent chapters. First we define the notion of free names in a process:

Definition 3.3 (Free names) *Let $P \in \mathsf{Proc}$, the set of* free names *in P, denoted $fn(P)$, is defined by the following:*

$$\begin{array}{rclrcl} fn(\mathbf{0}) & = & \emptyset & fn((\nu_\mu n)P) & = & fn(P) \setminus \{n\} \\ fn(P \mid Q) & = & fn(P) \cup fn(Q) & fn(!P) & = & fn(P) \\ fn(n^{\ell^a_n}[P]) & = & \{n\} \cup fn(P) & fn(in^{\ell^t} n.P) & = & \{n\} \cup fn(P) \\ fn(out^{\ell^t} n.P) & = & \{n\} \cup fn(P) & fn(open^{\ell^t} n.P) & = & \{n\} \cup fn(P) \end{array}$$

The above definition formalises that the free names of a process are those that do not occur within the scope of a restriction operator. Names that do occur within the scope of a restriction operator are called *bound names*. Thus it is possible for a name to be both free and bound (for different occurrences), e.g. the name n in the following expression

$$n^{\ell_1^a}[P] \mid (\nu_\mu n) n^{\ell_2^a}[Q]$$

A process, P, is *closed* if and only if $fn(P) = \emptyset$. Note that n is the only free name of the term in Example 3.1. We now turn to the definition of α-substitution:

Definition 3.4 (α-substitution) *Let $P \in \textsf{Proc}$ and $m, n \in \textsf{Nam}$. Denote by $P\{n \leftarrow m\}$ the result of α-substituting m for n in P, defined as follows:*

$$\begin{aligned}
\mathbf{0}\{n \leftarrow m\} &= \mathbf{0} \\
((\nu_\mu k)P)\{n \leftarrow m\} &= \begin{cases} (\nu_\mu k)P & \text{if } k = n \\ (\nu_\mu k)(P\{n \leftarrow m\}) & \text{if } k \neq n, k \neq m \\ (\nu_\mu k)((P\{m \leftarrow z\})\{n \leftarrow m\}) & \text{if } k \neq n, k = m, \\ & z \notin fn(P) \cup \{m, n\} \end{cases} \\
(P \mid Q)\{n \leftarrow m\} &= P\{n \leftarrow m\} \mid Q\{n \leftarrow m\} \\
(!P)\{n \leftarrow m\} &= !(P\{n \leftarrow m\}) \\
k^{\ell_k^a}[P]\{n \leftarrow m\} &= \begin{cases} m^{\ell_k^a}[P\{n \leftarrow m\}] & \text{if } k = n \\ k^{\ell_k^a}[P\{n \leftarrow m\}] & \text{if } k \neq n \end{cases} \\
(in^{\ell^t} k.P)\{n \leftarrow m\} &= \begin{cases} in^{\ell^t} m.(P\{n \leftarrow m\}) & \text{if } k = n \\ in^{\ell^t} k.(P\{n \leftarrow m\}) & \text{if } k \neq n \end{cases} \\
(out^{\ell^t} k.P)\{n \leftarrow m\} &= \begin{cases} out^{\ell^t} m.(P\{n \leftarrow m\}) & \text{if } k = n \\ out^{\ell^t} k.(P\{n \leftarrow m\}) & \text{if } k \neq n \end{cases} \\
(open^{\ell^t} k.P)\{n \leftarrow m\} &= \begin{cases} open^{\ell^t} m.(P\{n \leftarrow m\}) & \text{if } k = n \\ open^{\ell^t} k.(P\{n \leftarrow m\}) & \text{if } k \neq n \end{cases}
\end{aligned}$$

Following [CG98, CG97a] we *identify* terms up to renaming of bound variables:

$$(\nu_\mu n)P = (\nu_\mu m)(P\{n \leftarrow m\}) \qquad m \notin fn(P)$$

where $m, n \in \textsf{Nam}$, $\mu \in \textsf{Mar}$ and $P \in \textsf{Proc}$. Clearly, with this identification, we have to be careful in what follows to ensure that we handle identical terms correctly. It is particularly important to keep this identification in mind when defining the analysis to ensure that it handles identity correctly, i.e. ensures that identical terms are analysed identically.

3.3 Semantics

The semantics of the calculus is defined by a number of *reduction rules*. In order to minimise the number of these rules and make it easier to reason about the calculus and its semantics, a number of desirable properties, e.g., associativity of parallel composition, are explicitly incorporated in the semantics. This is done using a congruence on terms defined in Figure 3.1. Allowing for changes related to the modified syntax, the definition in Figure 3.1 is almost exactly as given in

(≡ Refl) $$\frac{}{P \equiv P}$$

(≡ Symm) $$\frac{P \equiv Q}{Q \equiv P}$$

(≡ Trans) $$\frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

(≡ Action) $$\frac{P \equiv Q}{M^{\ell^t}.P \equiv M^{\ell^t}.Q}$$

(≡ Repl) $$\frac{P \equiv Q}{!P \equiv !Q}$$

(≡ Amb) $$\frac{P \equiv Q}{n^{\ell^a}[P] \equiv n^{\ell^a}[Q]}$$

(≡ Par) $$\frac{P \equiv Q}{P \mid R \equiv Q \mid R}$$

(≡ Par Comm) $$\frac{}{P \mid Q \equiv Q \mid P}$$

(≡ Zero Par) $$\frac{}{P \mid \mathbf{0} \equiv P}$$

(≡ Res) $$\frac{P \equiv Q}{(\nu_\mu n)P \equiv (\nu_\mu n)Q}$$

(≡ Zero Repl) $$\frac{}{!\mathbf{0} \equiv \mathbf{0}}$$

(≡ Repl Par) $$\frac{}{!P \equiv P \mid !P}$$

(≡ Zero Res) $$\frac{}{(\nu_\mu n)\mathbf{0} \equiv \mathbf{0}}$$

(≡ Par Assoc) $$\frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)}$$

(≡ Res Res) $$\frac{}{(\nu_{\mu_1} n)(\nu_{\mu_2} m)P \equiv (\nu_{\mu_2} m)(\nu_{\mu_1} n)P} \quad (\text{if } n \neq m)$$

(≡ Res Par) $$\frac{}{(\nu_\mu n)(P \mid Q) \equiv P \mid (\nu_\mu n)Q} \quad (\text{if } n \notin fn(P))$$

(≡ Res Amb) $$\frac{}{(\nu_\mu n)m^{\ell^a}[P] \equiv m^{\ell^a}[(\nu_\mu n)P]} \quad (\text{if } n \neq m)$$

Figure 3.1: Structural Congruence on **Proc**

[CG97a], with the notable exception that we have added a sidecondition to the following rule:

$$(\equiv \text{Res Res}) \quad \frac{}{(\nu_{\mu_1} n)(\nu_{\mu_2} m)P \equiv (\nu_{\mu_2} m)(\nu_{\mu_1} n)P} \quad (\text{if } n \neq m)$$

The reason for this change is related to the discussion of the role of labels and markers in the previous section. In particular, the need for analysis information to remain invariant is of relevance here.

Having defined the congruence it is straightforward to define the reduction rules that make up the semantics:

$$(\text{Red In}) \quad \frac{}{n^{\ell^a_n}[in^{\ell^t} m.P \mid Q] \mid m^{\ell^a_m}[R] \to m^{\ell^a_m}[n^{\ell^a_n}[P \mid Q] \mid R]}$$

$$(\text{Red Out}) \quad \frac{}{m^{\ell^a_m}[n^{\ell^a_n}[out^{\ell^t} m.P \mid Q] \mid R] \to n^{\ell^a_n}[P \mid Q] \mid m^{\ell^a_m}[R]}$$

$$(\text{Red Open}) \quad \frac{}{open^{\ell^t} n.P \mid n^{\ell^a_n}[Q] \to P \mid Q} \qquad (\text{Red Amb}) \quad \frac{P \to Q}{n^{\ell^a_n}[P] \to n^{\ell^a_n}[Q]}$$

$$(\text{Red Res}) \quad \frac{P \to Q}{(\nu_\mu n)P \to (\nu_\mu n)Q} \qquad (\text{Red Par}) \quad \frac{P \to Q}{P \mid R \to Q \mid R}$$

$$(\text{Red} \equiv) \quad \frac{P \equiv P' \qquad P \to Q \qquad Q \equiv Q'}{P' \to Q'}$$

Again, the above definition differs from the one given in [CG97a, Section 2] only in that it is based on the modified syntax. In what follows we shall write $\to^*$ for the reflexive transitive closure of $\to$.

The rules (*Red Amb*), (*Red Res*) and (*Red Par*) enable reduction inside ambients, binding operators and in parallel composition respectively. The (*Red* $\equiv$) rule incorporates the congruence into the semantics. In order to move ambients around, the rules (*Red In*) and (*Red Out*) are used. Finally (*Red Open*) dissolves an ambient, thus allowing its content to spill into the immediate surroundings. In Figure 3.2 the basic rules are illustrated.

By inspecting the above reduction rules, it should be obvious that for any reduction (in one step) only *one* of the axioms[1] are used in the derivation of that reduction:

Fact 3.5 *Let $P, Q \in$* **Proc** *such that $P \to Q$. Only one of the reduction rules (axioms): (Red In), (Red Out) and (Red Open) are used in the derivation of $P \to Q$.*

[1]Rules with no premise are called axioms. In the above (*Red In*), (*Red Out*) and (*Red Open*), are axioms.

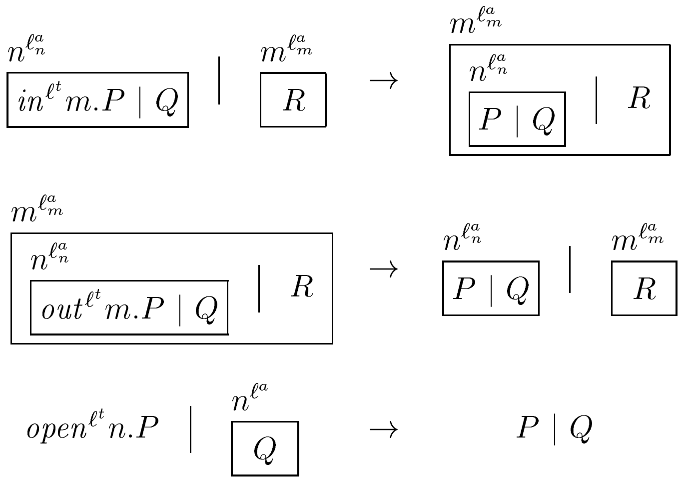


Figure 3.2: Pictorial representation of the basic reduction rules

Proof. By induction on the length of the derivation. ∎

This leads us to the definition of base steps:

Definition 3.6 (Base step) *Let* $P, Q \in \textsf{Proc}$ *such that* $P \rightarrow Q$. *The axiom used in the derivation of* $P \rightarrow Q$ *is called the* base step *of the reduction* $P \rightarrow Q$.

Remark 3.7 When creating and defining a process algebra, once the syntax and semantics is in place, it is not uncommon to define and study notions of equivalence, e.g., observational equivalence and contextual equivalence. These notions make it possible to abstract away irrelevant details, such as (minor) syntactic differences, and concentrate on the "relevant" issues. This has also been done for the ambient calculus: in [CG97b] Cardelli and Gordon recast a notion of observational equivalence from the π-calculus in the setting of mobile ambients.

Specifically, they define two notions of equivalence: bisimilarity and contextual equivalence, denoted by '$\sim$' and '$\simeq$' respectively. Informally two processes are bisimilar if and only if every move or action made by one process can be "simulated" by the other and vice versa; two processes are contextually equivalent if and only if their observable "behaviour" is indistinguishable in every conceivable context. We shall have no further need for these notions (apart from a few informal references) and thus we shall not go into further detail, merely refer to [CG97b] for the development and technical details of these relations.

In the recent paper [GC99] a notion of behavioural equivalence, in the form of a contextual equivalence, is studied. Here the contextual equivalence is defined in terms of observing whether or not an ambient, whose name is not restricted, is present at the top-level of the process at hand. Furthermore, techniques for proving (or disproving) processes equivalent are developed. ◇

Example 3.8 (Example 3.1 continued) Consider again the term from Example 3.1, we can apply the reduction rules in the following way

$$\begin{array}{rcl}(\nu_\mu m)(n^{\ell^a_n}[in^{\ell^t_1}m] \mid m^{\ell^a_m}[open^{\ell^t_2}n]) & \to & (\nu_\mu m)(m^{\ell^a_m}[open^{\ell^t_2}n \mid n^{\ell^a_n}[\mathbf{0}]]) \\ & \to & (\nu_\mu m)m^{\ell^a_m}[\mathbf{0}] \\ & \simeq & \mathbf{0}\end{array}$$

In the above, '$\simeq$' is the contextual equivalence described in Remark 3.7. Intuitively, a process that is $\simeq$-equivalent to $\mathbf{0}$ does not interact in any way with other ambients (or threads) around or outside it. Ambients (and threads) internal to such a process may, however, still interact with each other, though they can never leave the process. The final "$\simeq \mathbf{0}$" in the above follows as a special case of "The Perfect Firewall Equation" also defined in [CG97b]. $\diamond$

Example 3.9 (Example 3.2 continued) Reconsider the term modelling an agent trying to gain access to a firewall from Example 3.2:

$$\begin{array}{l}Agent \mid Firewall \equiv \\ \quad k_2^{\ell^a_{k_2}}[open^{\ell^t_6}k_1.k_3^{\ell^a_{k_3}}[Q]] \mid \\ \quad (\nu_\mu w)w^{\ell^a_w}[k_1^{\ell^a_{k_1}}[out^{\ell^t_1}w.in^{\ell^t_2}k_2.in^{\ell^t_3}w] \mid open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P]\end{array}$$

Applying the reduction rules of the semantics to this term, we obtain the following reduction sequence:

$$\begin{array}{ll}\multicolumn{2}{l}{k_2^{\ell^a_{k_2}}[open^{\ell^t_6}k_1.k_3^{\ell^a_{k_3}}[Q]] \mid (\nu_\mu w)w^{\ell^a_w}[k_1^{\ell^a_{k_1}}[out^{\ell^t_1}w.in^{\ell^t_2}k_2.in^{\ell^t_3}w] \mid open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P]} \\ \to^* & k_2^{\ell^a_{k_2}}[open^{\ell^t_6}k_1.k_3^{\ell^a_{k_3}}[Q] \mid k_1^{\ell^a_{k_1}}[in^{\ell^t_3}w]] \mid (\nu_\mu w)w^{\ell^a_w}[open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P] \\ \to & k_2^{\ell^a_{k_2}}[k_3^{\ell^a_{k_3}}[Q] \mid in^{\ell^t_3}w] \mid (\nu_\mu w)w^{\ell^a_w}[open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P] \\ \to & k_2^{\ell^a_{k_2}}[k_3^{\ell^a_{k_3}}[Q] \mid in^{\ell^t_3}w] \mid (\nu_\mu w)w^{\ell^a_w}[open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P] \\ \to & (\nu_\mu w)w^{\ell^a_w}[k_2^{\ell^a_{k_2}}[k_3^{\ell^a_{k_3}}[Q]] \mid open^{\ell^t_4}k_2.open^{\ell^t_5}k_3.P] \\ \to^* & (\nu_\mu w)w^{\ell^a_w}[Q \mid P]\end{array}$$

Thus the agent was successful in gaining access to the firewall and delivering the process, Q, for further computations. $\diamond$

Chapter 4

The Analysis

The aim of our analysis is to obtain information about the movement of the ambients and capabilities inside a given process. For each ambient in the process an acceptable analysis should give a conservative estimate of the set of ambients and capabilities that may reside in the ambient in a possible derivation sequence. This information can for example be used to check that only a certain set of ambients can enter a "protected" ambient. In Chapter 5 we elaborate on the possible applications of the analysis.

As explained in Chapter 3 we cannot use names in the analysis due to α-renaming so we use labels and markers instead. The analyses are pairs (I, H) of sets of ordered pairs:

$$\begin{array}{c} I \in \textsf{InAmb} = \mathcal{P}(\textsf{Lab}^a \times (\textsf{Lab}^a \cup \textsf{Lab}^t)) \\ H \in \textsf{HNam} = \mathcal{P}(\textsf{Lab}^a \times \textsf{Mar}) \end{array}$$

The I-component is the most interesting part of the analysis, since it provides information on which ambients and capabilities (represented by ambient labels and thread labels respectively) are inside which ambients. Intuitively, if an ambient label or thread label, say $\ell^{at} \in \textsf{Lab}$, resides in an ambient labelled ℓ^a in a possible derivation sequence, then $(\ell^a, \ell^{at}) \in I$ should hold for an analysis to be acceptable. The H-component of an analysis is responsible for recording the association between restrictions (represented by markers) and ambients (represented by ambient labels), i.e. if an ambient labelled ℓ^a is bound at a restriction marked μ then $(\ell^a, \mu) \in H$ should hold.

Example 4.1 For (I, H) to be an acceptable analysis of

$$(\nu_\mu m)(n^{\ell^a_n}[in^{\ell^t_1} m] \mid m^{\ell^a_m}[open^{\ell^t_2} n])$$

we require (among other things) that $(\ell^a_n, \ell^t_1) \in I$ and $(\ell^a_m, \ell^t_2) \in I$ since the capability $in^{\ell^t_1} m$ is inside the ambient n labelled ℓ^a_n and the capability $open^{\ell^t_2} n$ is inside the ambient m labelled ℓ^a_m. We depict this as:

$$\begin{tikzcd} & \ell^a_n \arrow[dl, "I"'] & & \ell^a_m \arrow[dl, "I"'] \\ \ell^t_1 & & \ell^t_2 & \end{tikzcd}$$

Furthermore we require that $(\ell^a_m, \mu) \in H$ since the ambient m labelled ℓ^a_m is bound at the restriction with marker μ. $\diamond$

We would like to compare the precision of different analyses and do so by imposing an ordering on $\mathsf{\textbf{InAmb}} \times \mathsf{\textbf{HNam}}$:

$$(I_1, H_1) \sqsubseteq (I_2, H_2) \text{ iff } I_1 \subseteq I_2 \ \wedge \ H_1 \subseteq H_2$$

In this ordering the smaller an analysis is the more precise it is. Note that $(\mathsf{\textbf{InAmb}} \times \mathsf{\textbf{HNam}}, \sqsubseteq)$ is a complete lattice (cf. Definition 4.6) with the least upper bound operator $\sqcup$ given by

$$(I_1, H_1) \sqcup (I_2, H_2) = (I_1 \cup I_2, H_1 \cup H_2)$$

We will use $\sqcup$ to combine the information from two analyses.

4.1 Explanation of the analysis

We want to check whether a proposed analysis, (I, H), is an acceptable analysis for a process, P. In order to keep track of the association between free names and markers we introduce the notion of a marker environment, $me : \mathsf{\textbf{Nam}} \rightarrow_{\text{fin}} \mathsf{\textbf{Mar}}$. For instance, if we want to check that a given analysis is acceptable for the process, $(\nu_{\mu_1} n)(m^{\ell^a_m}[P_1] \mid (\nu_{\mu_2} n)P_2)$, P_1 should be analysed in the context of a marker environment, me_1, where $me_1(n) = \mu_1$ and P_2 should be analysed in the context of a marker environment, me_2, where $me_2(n) = \mu_2$. We use a marker environment because terms are identified up to renaming of bound names and hence we cannot use the bound names themselves.

For the formulation of the analysis we shall write

$$\beta^{\ell^a}_{me}(P) \sqsubseteq (I, H) \ \wedge \ (I, H) \models_{me} P$$

when (I, H) is an acceptable analysis for P in the context of an enclosing ambient label ℓ^a and an initial marker environment, me. It is required that the free names of P are bound in the initial marker environment, i.e. $fn(P) \subseteq \mathrm{dom}(me)$. We will call β the representation function; it is specified in Table 4.1 and explained below. We say that if $(I, H) \models_{me} P$ then (I, H) fulfills the closure property with respect to P and me. The specification of $\models$ is given in Table 4.2 and explained below. The specifications of β and $\models$ are syntax directed, so both β and $\models$ are well defined.

$$
\begin{array}{lll}
(res) & \beta^{\ell^a}_{me}((\nu_\mu n)P) & = \beta^{\ell^a}_{me[n\mapsto\mu]}(P) \\
(zero) & \beta^{\ell^a}_{me}(\mathbf{0}) & = (\emptyset, \emptyset) \\
(par) & \beta^{\ell^a}_{me}(P \mid Q) & = \beta^{\ell^a}_{me}(P) \sqcup \beta^{\ell^a}_{me}(Q) \\
(repl) & \beta^{\ell^a}_{me}(!P) & = \beta^{\ell^a}_{me}(P) \\
(amb) & \beta^{\ell^a}_{me}(n^{\ell^a_n}[P]) & = \beta^{\ell^a_n}_{me}(P) \sqcup (\{(\ell^a, \ell^a_n)\}, \{(\ell^a_n, me(n))\}) \\
(in) & \beta^{\ell^a}_{me}(in^{\ell^t} n.P) & = \beta^{\ell^a}_{me}(P) \sqcup (\{(\ell^a, \ell^t)\}, \emptyset) \\
(out) & \beta^{\ell^a}_{me}(out^{\ell^t} n.P) & = \beta^{\ell^a}_{me}(P) \sqcup (\{(\ell^a, \ell^t)\}, \emptyset) \\
(open) & \beta^{\ell^a}_{me}(open^{\ell^t} n.P) & = \beta^{\ell^a}_{me}(P) \sqcup (\{(\ell^a, \ell^t)\}, \emptyset)
\end{array}
$$

Table 4.1: The representation function

Intuitively, the representation function, β, gives the most precise information for a process without taking into account the reductions that the process might perform. In other words, $\beta^{\ell^a}_{me}(P)$ represents the initial state of P where the nearest enclosing ambient of P is ℓ^a. The individual clauses of Table 4.1 is explained in the following.

In (*res*) the marker environment is updated to reflect the fact that the free occurences of n in P is bound at a restriction marked μ.

As $\mathbf{0}$ does not contain any labels its state is $(\emptyset, \emptyset)$ in (*zero*).

In (*par*) there are two "recursive calls", which are combined using $\sqcup$ in order to combine the states from the two subprocesses.

In (*repl*) we see that the state of $!P$ is identical to the state of P; this is justified by the fact that the congruence rule for $!P$ states that $!P \equiv !P \mid P$, i.e. $!P$ may turn into any finite number of *exact* copies of P where the markers and labels are the same in the different copies.

In (*amb*) we record that the ambient is inside its enclosing ambient and that the ambient is bound at a restriction marked μ; furthermore the state of the subprocess, P, is computed with the information that the label of its enclosing ambient is ℓ^a_n.

In the clause for (*in*) it is recorded that the capability is inside its enclosing ambient. (*out*) and (*open*) are similar to (*in*).

Intuitively, the closure property, $\models$, "simulates" the reductions that the process might perform when the information from a proposed analysis is taken into account. The specification of $\models$ from Table 4.2 follows the same outline as β.

$$
\begin{array}{lll}
(res) & (I,H) \models_{me} (\nu_\mu n)P & \text{iff} \quad (I,H) \models_{me[n\mapsto\mu]} P \\[1ex]
(zero) & (I,H) \models_{me} \mathbf{0} & \text{always} \\[1ex]
(par) & (I,H) \models_{me} P \mid Q & \text{iff} \quad (I,H) \models_{me} P \;\wedge\; (I,H) \models_{me} Q \\[1ex]
(repl) & (I,H) \models_{me} !P & \text{iff} \quad (I,H) \models_{me} P \\[1ex]
(amb) & (I,H) \models_{me} n^{\ell^a}[P] & \text{iff} \quad (I,H) \models_{me} P \\[1ex]
(in) & (I,H) \models_{me} in^{\ell^t} n.P & \text{iff} \quad (I,H) \models_{me} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a : ((\ell^a, \ell^t) \in I \;\wedge\; (\ell^{a''}, \ell^a) \in I \;\wedge\; (\ell^{a''}, \ell^{a'}) \in I} \\
& \multicolumn{2}{l}{\qquad \wedge\; (\ell^{a'}, me(n)) \in H) \;\Rightarrow\; (\ell^{a'}, \ell^a) \in I} \\[1ex]
(out) & (I,H) \models_{me} out^{\ell^t} n.P & \text{iff} \quad (I,H) \models_{me} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a : ((\ell^a, \ell^t) \in I \;\wedge\; (\ell^{a'}, \ell^a) \in I \;\wedge\; (\ell^{a''}, \ell^{a'}) \in I} \\
& \multicolumn{2}{l}{\qquad \wedge\; (\ell^{a'}, me(n)) \in H) \;\Rightarrow\; (\ell^{a''}, \ell^a) \in I} \\[1ex]
(open) & (I,H) \models_{me} open^{\ell^t} n.P & \text{iff} \quad (I,H) \models_{me} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a : ((\ell^a, \ell^t) \in I \;\wedge\; (\ell^a, \ell^{a'}) \in I \;\wedge\; (\ell^{a'}, me(n)) \in H)} \\
& \multicolumn{2}{l}{\qquad \Rightarrow\; \left\{(\ell^a, \ell') \mid (\ell^{a'}, \ell') \in I\right\} \subseteq I}
\end{array}
$$

Table 4.2: The closure property

In the clause (*res*) of Table 4.2 the marker environment is updated to reflect that the free occurrences of n in P is bound at the restriction marked μ.

The clause (*zero*) places no demands on the analysis as it does not involve any labels.

The clause (*par*) contains two "recursive calls" using the same annotation on the satisfaction symbol.

In the clause (*repl*) we see that the analysis ignores the fact that $!P$ may turn into several copies of P.

As an ambient boundary is not active in itself we see in (*amb*) that the closure property holds if it holds for the subprocess.

In the last three clauses we deal with capabilities, which are the complicated cases, since the reductions of a process always involves a capability (cf. the Definition 3.6 of base steps). In the clause (*in*) we first check that the closure property holds for the subprocess. The second conjunct is motivated by the following scenario:

$$p^{\ell^{a''}}[\cdots \overbrace{q^{\ell^a}[\cdots in^{\ell^t} n.P \cdots]} \cdots \mid n^{\ell^{a'}}[\ \cdots\ ]\cdots]$$

In order for (I, H) to be an acceptable analysis of the above process we should have that $(\ell^t, \ell^a) \in I, (\ell^{a''}, \ell^a) \in I, (\ell^{a''}, \ell^{a'}) \in I$ and $(\ell^{a'}, me(n)) \in H$. We depict this as

$$\begin{tikzcd}
& & \ell^{a''} \arrow[ddl, "I"'] \arrow[dr, "I"] & \\
& & & \ell^{a'}\rangle me(n) \arrow[dll, dashed, "I"] \\
& \ell^{a} \arrow[dl, "I"'] & & \\
\ell^{t} & & &
\end{tikzcd}$$

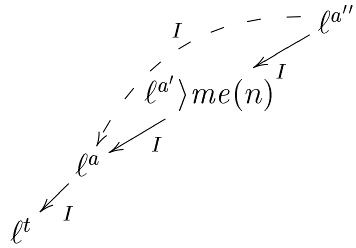
where $\ell^{a'}\rangle me(n)$ means that $(\ell^{a'}, me(n)) \in H$. The dashed line indicates that $(\ell^{a'}, \ell^a) \in I$ should hold since the ambient q labelled ℓ^a has the possibility of moving in the ambient n labelled $\ell^{a'}$.

The clauses (*out*) and (*open*) are similar to (*in*). In (*out*) the following scenario motivates the second conjunct:

$$p^{\ell^{a''}}[\ \cdots\ n^{\ell^{a'}}[\cdots \overbrace{q^{\ell^a}[\cdots out^{\ell^t} n.P \cdots]} \cdots]\ \cdots]$$

We ought to have that $(\ell^a, \ell^t) \in I, (\ell^{a'}, \ell^a) \in I, (\ell^{a''}, \ell^{a'}) \in I$ and $(\ell^{a'}, me(n)) \in H$

by an argument analogous to the above:

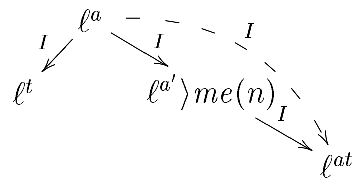


Again the dashed line indicates that $(\ell^{a'}, \ell^a) \in I$ is required, due to the fact that the ambient q labelled ℓ^a has the possibility of moving out of the ambient n labelled $\ell^{a'}$ and thus become a child of the ambient p labelled $\ell^{a''}$.

In $(open)$ we have the following scenario leading to the second conjunct:

$$p^{\ell^a}[\ \cdots\ open^{\ell^t} n.P \cdots \mid n^{\ell^{a'}}[\ \overbrace{\cdots in^{\ell'^t}.Q \cdots q^{\ell^a_q}[\cdots]\cdots}\ ]\cdots]$$

We ought to have that $(\ell^a, \ell^t) \in I, (\ell^a, \ell^{a'}) \in I$ and $(\ell^{a'}, me(n)) \in H$:



where ℓ^{at} represents the labels of $\left\{\ell'^{at} \,\middle|\, (\ell^{a'}, \ell'^{at}) \in I\right\}$. In the above scenario ℓ'^t and ℓ^a_q ought to be in this set for an analysis to be acceptable, since the ambient n labelled $\ell^{a'}$ has the possibility of being opened, so that the capability $in^{\ell^t} m$ and the ambient q labelled ℓ^a_q "flows" into the ambient p labelled ℓ^a.

4.1.1 Observations on the analysis

In the terminology of data flow analysis as described in [NNH], we say that the analysis is flow-insensitive since it ignores the order in which two capabilities occur. To illustrate how this affects the precision of the analysis, we consider the process:

$$n^{\ell^a_n}[open^{\ell^t_1} q.in^{\ell^t_2} m] \mid m^{\ell^a_m}[\mathbf{0}]$$

All acceptable analyses reports that the ambient n can end up in the ambient m since the capability $in^{\ell^t_2} m$ occurs inside the ambient n, and n has the ambient m as a sibling. A flow-sensitive analysis could do better than this by determining that the capability $open^{\ell^t_1} q$ is not performed and then determines that it can refrain from analysing the capability $in^{\ell_2^t} m$. We conjecture that it would relatively easy to modify the present analysis so that it would not analyse $in^{\ell_2^t} m$.

Furthermore we say that the analysis is context-insensitive (or mono-variant) in the sense that a capability is analysed the same way regardless of the context in which it occurs.

4.1.2 Alternative specifications

The presented specification of when a proposed analysis is acceptable is the result of an iteration where several specifications have been considered. Here we will sketch some possible alternatives.

First of all there is a possibility to combine the representation function and the closure property in one predicate: $(I, H) \models_{me}^{\ell^a} P$. As an example the clause for (amb) would be

$$(I, H) \models_{me}^{\ell^a} n^{\ell_n^a}[P] \text{ iff } (I, H) \models_{me}^{\ell_n^a} P \;\wedge\; (\ell^a, \ell_n^a) \in I \;\wedge\; (\ell_n^a, me(n)) \in H$$

This is the approach taken in [NNHJ99]. The advantage of this is a more compact specification, but we feel that the separation of the analysis into a representation function and a closure property provides better insight into the analysis by separating the static property of a process (representation function) from the dynamic property (closure property).

We have also studied the possibility of specifying the analysis by "looking backwards", i.e. when a capability is analysed the preceding capability is known, and one can then use the information of where this capability can be. The disadvantage of this approach is a more complex specification where there is no easy way of making a division into a representation function and a closure property since a proposed analysis does not provide information on the relative position of the capabilities inside an ambient.

4.2 Properties of the analysis

In this section we investigate some of the formal properties of the analysis described in the previous section. In particular we establish the three main results: identical terms are analysed identically, semantic correctness of the analysis and the existence of least/best analyses.

4.2.1 Preliminaries

In anticipation of the following sections, we briefly review the formal notions used: basic definitions from order and lattice theory. Since the definitions are standard we shall not go into details or further explanations; for a thorough treatment see [DP90].

Definition 4.2 (Partial Order) *Let P be a set and let $\sqsubseteq$ be a binary relation on P. Then $(P, \sqsubseteq)$ is a* partial order *if and only if*

1. $\forall a \in P : a \sqsubseteq a$ *(reflexivity)*
2. $\forall a, b, c \in P : a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$ *(transitivity)*
3. $\forall a, b \in P : a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$ *(anti-symmetry)*

If P has a least element, i.e. an element $l \in P$ such that $\forall p \in P : l \sqsubseteq p$, it is denoted $\perp$. Analogously, the greatest element, if it exists, is denoted $\top$. Trivially both $\perp$ and $\top$ are unique if they exist.

Generalising the notions of least and greatest element we arrive at the following:

Definition 4.3 (Upper Bounds) *Let $(P, \sqsubseteq)$ be a partial order set and let $S \subseteq P$, then $u \in P$ is said to be an* upper bound *of S if $\forall s \in S : s \sqsubseteq u$. Furthermore, the upper bound u is called the* least upper bound *of S if $u \sqsubseteq x$ for all upper bounds, x of S.*

The least upper bound of $S \subseteq P$ is denoted $\bigsqcup S$ whenever it exists. We write $x \sqcup y$ for $\bigsqcup \{x, y\}$. Dually, we define:

Definition 4.4 (Lower bounds) *Let $(P, \sqsubseteq)$ be a partial order set and let $S \subseteq P$, then $l \in P$ is said to be a* lower bound *of S if $\forall s \in S : l \sqsubseteq s$. Furthermore, the lower bound l is called the* greatest lower bound *of S if $x \sqsubseteq l$ for all lower bounds, x of S.*

The greatest lower bound of $S \subseteq P$ is denoted $\sqcap S$ whenever it exists. We write $x \sqcap y$ for $\sqcap \{x, y\}$.

With the above definitions in place, the notion of a lattice can now be defined:

Definition 4.5 (Lattice) *Let $(P, \sqsubseteq)$ be a partial order set, such that $P \neq \emptyset$, then P is called a* lattice *if $x \sqcup y$ and $x \sqcap y$ exist for all $x, y \in P$.*

An important special instance of a lattice is that of a complete lattice:

Definition 4.6 (Complete Lattice) *Let $(P, \sqsubseteq)$ be a partial order, then P is called a* complete lattice *if $\bigsqcup S$ and $\sqcap S$ exist for all $S \subseteq P$.*

The following definition of Moore family plays an important part in establishing the existence of (least) analyses for the specification in Section 4.1.

Definition 4.7 (Moore family) *Let $L = (L, \sqsubseteq)$ be a complete lattice, and let $S \subseteq L$ then S is a* Moore family *if and only if $\forall S' \subseteq S : (\sqcap S') \in S$*

Excepting Definition 4.7, the above definitions follow [DP90] rather closely.

4.2.2 Analysis of identical terms

As noted in Chapter 3, α-equivalent terms are considered to be identical. We need to establish that the analysis treats identical terms identically. This is done in two stages: First it is shown that '$\equiv$'-equivalent terms are treated the same, and next that also α-equivalent, i.e. identical, terms are treated identically.

We start off by proving an auxiliary result, showing that the representation function depends only on the free variables in the given process:

Fact 4.8 *Let $P \in$ **Proc**, $\ell \in$ **Lab**^{a} and $me \in$ **MEnv** then for $n \notin fn(P)$:*

$$\beta^{\ell}_{me}(P) = \beta^{\ell}_{me[n \mapsto \mu]}(P)$$

Proof. By structural induction on P.

Case ($m^{\ell^a_m}[P]$): Since $m \in fn(m^{\ell^a_m}[P])$ we must have that $m \neq n$. Furthermore we have:

$$\begin{array}{rcl}
\beta^{\ell}_{me}(m^{\ell^a_m}[P]) & = & \beta^{\ell^a}_{me}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, me(m))\}) \\
& = & \beta^{\ell^a}_{me[n \mapsto \mu]}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, me(m))\}) \\
& = & \beta^{\ell^a}_{me[n \mapsto \mu]}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, me[n \mapsto \mu](m))\}) \\
& = & \beta^{\ell}_{me[n \mapsto \mu]}(m^{\ell^a_m}[P])
\end{array}$$

Case ($(\nu_{\mu'} m)P$): There are two subcases: $m = n$ and $m \neq n$:

Case ($m = n$): We proceed by simple calculation:

$$\begin{array}{rcl}
\beta^{\ell}_{me}((\nu_{\mu'} m)P) & = & \beta^{\ell}_{me[m \mapsto \mu']}(P) \\
& = & \beta^{\ell}_{me[n \mapsto \mu, m \mapsto \mu']}(P) \\
& = & \beta^{\ell}_{me[n \mapsto \mu]}((\nu_{\mu'} m)P)
\end{array}$$

Case ($m \neq n$): Again we proceed by simple calculation:

$$\begin{array}{rcl}
\beta^{\ell}_{me}((\nu_{\mu'} m)P) & = & \beta^{\ell}_{me[m \mapsto \mu']}(P) \\
& = & \beta^{\ell}_{me[m \mapsto \mu', n \mapsto \mu]}(P) \\
& = & \beta^{\ell}_{me[n \mapsto \mu, m \mapsto \mu']}(P) \\
& = & \beta^{\ell}_{me[n \mapsto \mu]}((\nu_{\mu'} m)P)
\end{array}$$

Where we use that $m \neq n \wedge n \notin fn((\nu_{\mu'} m)P) \Rightarrow n \notin fn(P)$ in order to apply the induction hypothesis.

The remaining cases are trivial. ∎

Using the above fact, we now prove that the representation function is invariant with respect to '$\equiv$'-equivalence.

Fact 4.9 *Let* $P, Q \in \textsf{Proc}$ *then* $\forall \ell \in \textsf{Lab}^a, \forall me \in \textsf{MEnv}$ *we have*

$$P \equiv Q \Rightarrow \beta^{\ell}_{me}(P) = \beta^{\ell}_{me}(Q)$$

Proof. By induction on the shape of the derivation of $P \equiv Q$.

Case ($\equiv$ Res Par): We have that $(\nu_\mu n)(P \mid Q) \equiv P \mid (\nu_\mu n)Q$ and that $n \notin fn(P)$. Now we have the following:

$$\begin{array}{rcl}
\beta^{\ell}_{me}((\nu_\mu n)P \mid Q) & = & \beta^{\ell}_{me[n\mapsto\mu]}(P \mid Q) \\
& = & \beta^{\ell}_{me[n\mapsto\mu]}(P) \sqcup \beta^{\ell}_{me[n\mapsto\mu]}(Q) \\
& = & \beta^{\ell}_{me}(P) \sqcup \beta^{\ell}_{me[n\mapsto\mu]}(Q) \\
& = & \beta^{\ell}_{me}(P) \sqcup \beta^{\ell}_{me}((\nu_\mu n)Q) \\
& = & \beta^{\ell}_{me}(P \mid (\nu_\mu n)Q)
\end{array}$$

Case ($\equiv$ Res Amb): Assuming that $(\nu_\mu n)m^{\ell^a_m}[P] \equiv m^{\ell^a_m}[(\nu_\mu n)P]$ where $n \neq m$, we have that:

$$\begin{array}{rcl}
\beta^{\ell}_{me}((\nu_\mu n)m^{\ell^a_m}[P]) & = & \beta^{\ell}_{me[n\mapsto\mu]}(m^{\ell^a_m}[P]) \\
& = & \beta^{\ell^a}_{me[n\mapsto\mu]}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, me[n \mapsto \mu](m))\}) \\
& = & \beta^{\ell^a}_{me[n\mapsto\mu]}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, me(m))\}) \\
& = & \beta^{\ell}_{me}(m^{\ell^a_m}[(\nu_\mu n)P])
\end{array}$$

The remaining cases are either trivial or similar to the above. ∎

Having established that the representation function, β, is invariant with respect to '$\equiv$'-equivalent terms, we proceed to prove that the same holds true for the closure property. First a helpful auxiliary result:

Fact 4.10 *Let* $P \in \textsf{Proc}$ *then if* $n \notin fn(P)$*:*

$$(I, H) \models_{me} P \iff (I, H) \models_{me[n\mapsto\mu]} P$$

Proof. By structural induction on P.

Case ($(\nu_{\mu'} m)P$): There are two subcases $m = n$ and $m \neq n$:

Case ($m = n$):

$$\begin{array}{rcl}
(I, H) \models_{me} (\nu_{\mu'} m)P & \iff & (I, H) \models_{me[m\mapsto\mu']} P \\
& \iff & (I, H) \models_{me[n\mapsto\mu, m\mapsto\mu']} P \\
& \iff & (I, H) \models_{me[n\mapsto\mu]} (\nu_{\mu'} m)P
\end{array}$$

Case ($m \neq n$):

$$\begin{array}{rcl}
(I, H) \models_{me} (\nu_{\mu'} m)P & \iff & (I, H) \models_{me[m\mapsto\mu']} P \\
& \iff & (I, H) \models_{me[m\mapsto\mu', n\mapsto\mu]} P \\
& \iff & (I, H) \models_{me[n\mapsto\mu, m\mapsto\mu']} P \\
& \iff & (I, H) \models_{me[n\mapsto\mu]} (\nu_{\mu'} m)P
\end{array}$$

Using that $m \neq n \wedge n \notin fn((\nu_{\mu'} m)P) \Rightarrow n \notin fn(P)$.

Case ($in^{\ell^t} m.P$): Note that since $n \notin fn(in^{\ell^t} m.P)$ we have that $m \neq n$. Furthermore:

$$
\begin{array}{ll}
(I,H) \models_{me} in^{\ell^t} m.P & \\
\quad \iff & (I,H) \models_{me} P \wedge \\
& (\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : \\
& \quad ((\ell^a, \ell^t) \in I \wedge (\ell^{a''}, \ell^a) \in I \wedge (\ell^{a''}, \ell^{a'}) \in I \wedge \\
& \quad (\ell^{a'}, me(m)) \in H) \Rightarrow (\ell^{a'}, \ell^a) \in I) \\
\quad \iff & (I,H) \models_{me[n \mapsto \mu]} P \wedge \\
& (\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : \\
& \quad ((\ell^a, \ell^t) \in I \wedge (\ell^{a''}, \ell^a) \in I \wedge (\ell^{a''}, \ell^{a'}) \in I \wedge \\
& \quad (\ell^{a'}, me[n \mapsto \mu](m)) \in H) \Rightarrow (\ell^{a'}, \ell^a) \in I) \\
\quad \iff & (I,H) \models_{me[n \mapsto \mu]} in^{\ell^t} m.P
\end{array}
$$

The remaining cases are analogous to the above or trivial. ∎

Using the above fact in conjunction with Fact 4.8 the Corollary below can be established, showing that the representation function and the closure property depends only on the free names occuring in the process that is analysed:

Corollary 4.11 *Let $P \in \mathbf{Proc}$, $\ell \in \mathbf{Lab}^a$ and $me, me' \in \mathbf{MEnv}$. Now if $\forall n \in fn(P) : me(n) = me'(n)$ then*

1. $\beta^{\ell}_{me}(P) = \beta^{\ell}_{me'}(P)$
2. $(I,H) \models_{me} P \iff (I,H) \models_{me'} P$

Proof. Follows from Fact 4.8 and Fact 4.10. ∎

We are now in a position to prove that the analysis is invariant with respect to '≡'-equivalent terms:

Fact 4.12 *Let $P \in \mathbf{Proc}$, $me \in \mathbf{MEnv}$ then*

$$P \equiv Q \Rightarrow ((I,H) \models_{me} P \iff (I,H) \models_{me} Q)$$

Proof. By induction on the derivation of $P \equiv Q$.

Case (≡ Repl Par): Assuming that $!P \equiv !Q$ because $P \equiv Q$, we have:

$$
\begin{array}{rcl}
(I,H) \models_{me} !P & \iff & (I,H) \models_{me} P \\
& \iff & (I,H) \models_{me} Q \\
& \iff & (I,H) \models_{me} !Q
\end{array}
$$

Case ($\equiv$ Res Par): Assume that $n \notin fn(P)$, then $(\nu_\mu n)(P \mid Q) \equiv P \mid (\nu_\mu n)Q$. Giving rise to the following:

$$
\begin{array}{ll}
(I,H) \models_{me} (\nu_\mu n)(P \mid Q) & \\
\quad \iff & (I,H) \models_{me[n\mapsto\mu]} P \mid Q \\
\quad \iff & (I,H) \models_{me[n\mapsto\mu]} P \wedge (I,H) \models_{me[n\mapsto\mu]} Q \\
\quad \iff & (I,H) \models_{me} P \wedge (I,H) \models_{me} (\nu_\mu n)Q \\
\quad \iff & (I,H) \models_{me} P \mid (\nu_\mu n)Q
\end{array}
$$

The remaining cases are similar or trivial. ∎

Thus we have established that the analysis is invariant under '$\equiv$'-equivalence.

With the aid of the following trivial fact, we now turn our attention towards proving that both the representation function and the analysis are invariant under α-equivalence.

Fact 4.13 *Let $P \in$ **Proc** then if $n \notin fn(P)$: $P \equiv (\nu_\mu n)P$*

Proof. Using definition of the '$\equiv$'-congruence we calculate the following:

$$
\begin{array}{rcl}
P \equiv \mathbf{0} \mid P \Rightarrow (\nu_\mu n)P & \equiv & (\nu_\mu n)\mathbf{0} \mid P \\
& \equiv & P \mid (\nu_\mu n)\mathbf{0} \\
& \equiv & P \mid \mathbf{0} \\
& \equiv & P
\end{array}
$$

This ends the proof. ∎

Next we prove that the representation function is invariant under α-equivalence:

Proposition 4.14 *Let $P \in$ **Proc**, $\ell \in$ **Lab**^{a} and $me \in$ **MEnv** then for $m \notin fn(P)$ the following holds:*

$$\beta^\ell_{me}((\nu_\mu n)P) = \beta^\ell_{me}((\nu_\mu m)(P\{n \leftarrow m\}))$$

Proof. By structural induction on P.

Case ($P \mid Q$):

$$
\begin{array}{rcl}
\beta^\ell_{me}((\nu_\mu n)(P \mid Q)) & = & \beta^\ell_{me[n\mapsto\mu]}(P \mid Q) \\
& = & \beta^\ell_{me[n\mapsto\mu]}(P) \sqcup \beta^\ell_{me[n\mapsto\mu]}(Q) \\
& = & \beta^\ell_{me[m\mapsto\mu]}(P\{n \leftarrow m\}) \sqcup \beta^\ell_{me[m\mapsto\mu]}(Q\{n \leftarrow m\}) \\
& = & \beta^\ell_{me[m\mapsto\mu]}(P\{n \leftarrow m\} \mid Q\{n \leftarrow m\}) \\
& = & \beta^\ell_{me[m\mapsto\mu]}((P \mid Q)\{n \leftarrow m\}) \\
& = & \beta^\ell_{me}((\nu_\mu m)((P \mid Q)\{n \leftarrow m\}))
\end{array}
$$

Case ($(\nu_{\mu'} k)P$): There are three subcases:

Case ($k = n$): We have $k = n$ and $m \notin fn(P)$ by assumption, now it follows from Fact 4.13 that

$$(\nu_\mu n)(\nu_{\mu'} k)P \equiv (\nu_{\mu'} k)P \equiv (\nu_\mu m)(\nu_{\mu'} k)P$$

thus from Fact 4.9 and since $(\nu_{\mu'} k)P = ((\nu_{\mu'} k)P)\{n \leftarrow m\}$ (because $k = n$) we then have

$$\begin{aligned} \beta^\ell_{me}((\nu_\mu n)(\nu_{\mu'} k)P) &= \beta^\ell_{me}((\nu_\mu m)(\nu_{\mu'} k)P) \\ &= \beta^\ell_{me}((\nu_\mu m)(((\nu_{\mu'} k)P)\{n \leftarrow m\})) \end{aligned}$$

Case ($k \neq n, k \neq m$): By straightforward calculation we obtain:

$$\begin{aligned} \beta^\ell_{me}((\nu_\mu n)(\nu_{\mu'} k)P) &= \beta^\ell_{me[n\mapsto\mu,k\mapsto\mu']}(P) \\ &= \beta^\ell_{me[k\mapsto\mu',n\mapsto\mu]}(P) \\ &= \beta^\ell_{me[k\mapsto\mu',m\mapsto\mu]}(P\{n \leftarrow m\}) \\ &= \beta^\ell_{me[m\mapsto\mu,k\mapsto\mu']}(P\{n \leftarrow m\}) \\ &= \beta^\ell_{me}((\nu_\mu m)(((\nu_{\mu'} k)P)\{n \leftarrow m\})) \end{aligned}$$

Case ($k \neq n, k = m$): First observe that for $z \in \textbf{Nam}$ such that $z \notin fn(P) \cup \{m, n\}$. It follows from the induction hypothesis and the above case that

$$\beta^\ell_{me'}((\nu_{\mu'} k)P) = \beta^\ell_{me'}((\nu_{\mu'} z)P\{k \leftarrow z\})$$

for some $\ell \in \textbf{Lab}^a$ and $me' \in \textbf{MEnv}$. We continue by the following calculation:

$$\begin{aligned} \beta^\ell_{me}((\nu_\mu n)(\nu_{\mu'} k)P) &= \beta^\ell_{me[n\mapsto\mu]}((\nu_{\mu'} z)P\{k \leftarrow z\}) \\ &= \beta^\ell_{me[n\mapsto\mu,z\mapsto\mu']}(P\{k \leftarrow z\}) \\ &= \beta^\ell_{me[z\mapsto\mu',n\mapsto\mu]}(P\{k \leftarrow z\}) \\ &= \beta^\ell_{me[z\mapsto\mu',m\mapsto\mu]}((P\{k \leftarrow z\})\{n \leftarrow m\}) \\ &= \beta^\ell_{me[m\mapsto\mu,z\mapsto\mu']}((P\{k \leftarrow z\})\{n \leftarrow m\}) \\ &= \beta^\ell_{me}((\nu_\mu m)(\nu_{\mu'} z)((P\{k \leftarrow z\})\{n \leftarrow m\})) \\ &= \beta^\ell_{me}((\nu_\mu m)(((\nu_{\mu'} k)P)\{n \leftarrow m\})) \end{aligned}$$

where the final equality follows from the definition of α-substitution, cf. Definition 3.4.

The remaining cases are similar or trivial. ■

Thus the representation function is invariant under α-equivalence. Finally we prove that this is also true of the closure property:

Proposition 4.15 *Let $P \in$ **Proc** and $me \in$ **MEnv**. For $m \notin fn(P)$:*

$$(I,H) \models_{me} (\nu_\mu n)P \iff (I,H) \models_{me} (\nu_{\mu'} m)P\{n \leftarrow m\}$$

Proof. By structural induction on P.

Case ($k^{\ell^a_k}[P]$): There are two subcases:

Case ($k = n$):

$$\begin{array}{ll} (I,H) \models_{me} (\nu_\mu n)k^{\ell^a_k}[P] & \\ \iff & (I,H) \models_{me[n\mapsto\mu]} k^{\ell^a_k}[P] \\ \iff & (I,H) \models_{me[n\mapsto\mu]} P \\ \iff & (I,H) \models_{me[m\mapsto\mu]} P\{n \leftarrow m\} \\ \iff & (I,H) \models_{me[m\mapsto\mu]} m^{\ell^a_m}[P\{n \leftarrow m\}] \\ \iff & (I,H) \models_{me[m\mapsto\mu]} (k^{\ell^a_k}[P])\{n \leftarrow m\} \\ \iff & (I,H) \models_{me} (\nu_\mu m)((k^{\ell^a_k}[P])\{n \leftarrow m\}) \end{array}$$

Case ($k \neq n$): Analogous to the above.

Case ($(\nu_{\mu'} k)P$): There are three subcases to consider:

Case ($k = n$): Using Fact 4.10 we obtain:

$$\begin{array}{ll} (I,H) \models_{me} (\nu_\mu n)(\nu_{\mu'} k)P & \\ \iff & (I,H) \models_{me[n\mapsto\mu]} (\nu_{\mu'} k)P \\ \iff & (I,H) \models_{me} (\nu_{\mu'} k)P \\ \iff & (I,H) \models_{me[m\mapsto\mu]} (\nu_{\mu'} k)P \\ \iff & (I,H) \models_{me[m\mapsto\mu]} ((\nu_{\mu'} k)P)\{n \leftarrow m\} \\ \iff & (I,H) \models_{me} (\nu_\mu m)(((\nu_{\mu'} k)P)\{n \leftarrow m\}) \end{array}$$

Case ($k \neq n, k \neq m$):

$$\begin{array}{ll} (I,H) \models_{me} (\nu_\mu n)(\nu_{\mu'} k)P & \\ \iff & (I,H) \models_{me[n\mapsto\mu,k\mapsto\mu']} P \\ \iff & (I,H) \models_{me[k\mapsto\mu',n\mapsto\mu]} P \\ \iff & (I,H) \models_{me[k\mapsto\mu',m\mapsto\mu]} P\{n \leftarrow m\} \\ \iff & (I,H) \models_{me[m\mapsto\mu,k\mapsto\mu']} P\{n \leftarrow m\} \\ \iff & (I,H) \models_{me} (\nu_{\mu'} m)(((\nu_{\mu'} k)P)\{n \leftarrow m\}) \end{array}$$

Case ($k \neq n, k = m$): Analogously to the corresponding case in the proof of Proposition 4.14, observe again that for $z \in$ **Nam** such that $z \notin fn(P) \cup \{m, n\}$, it follows from the induction hypothesis and the case above that:

$$(I,H) \models_{me'} (\nu_{\mu'} k)P \iff (I,H) \models_{me'} (\nu_{\mu'} z)P\{m \leftarrow z\}$$

for some $me' \in \mathsf{MEnv}$. Now using this observation we calculate:

$$\begin{array}{ll}(I,H) \models_{me} (\nu_\mu n)(\nu_{\mu'} k)P & \\ \iff & (I,H) \models_{me[n\mapsto\mu]} (\nu_{\mu'} z)P\{m \leftarrow z\} \\ \iff & (I,H) \models_{me[n\mapsto\mu, z\mapsto\mu']} P\{m \leftarrow z\} \\ \iff & (I,H) \models_{me[z\mapsto\mu', n\mapsto\mu]} P\{m \leftarrow z\} \\ \iff & (I,H) \models_{me[z\mapsto\mu', m\mapsto\mu]} (P\{m \leftarrow z\})\{n \leftarrow m\} \\ \iff & (I,H) \models_{me[m\mapsto\mu, z\mapsto\mu']} (P\{m \leftarrow z\})\{n \leftarrow m\} \\ \iff & (I,H) \models_{me[m\mapsto\mu]} (\nu_{\mu'} z)((P\{m \leftarrow z\})\{n \leftarrow m\}) \\ \iff & (I,H) \models_{me} (\nu_\mu m)(((\nu_{\mu'} k)P)\{n \leftarrow m\})\end{array}$$

The remaining cases are analogous or trivial. ■

In other words: Identical terms are analysed identically.

In summary we have established the invariance of both the representation function and the closure property under both '$\equiv$'-equivalence and α-equivalence.

4.2.3 Semantic Correctness

We now turn towards establishing the semantic correctness of the analysis. This is done by establishing a *subject reduction* property for the analysis, i.e. establishing that an acceptable analysis remains acceptable under evaluation.

We first establish a technical result needed to prove subject reduction:

Fact 4.16 *Let $\ell^a, \ell^{a'} \in \mathsf{Lab}^a$, $me \in \mathsf{MEnv}$, $P \in \mathsf{Proc}$ and $\beta^{\ell^a}_{me}(P) = (I', H')$ then we have that:*

$$((I', H') \sqsubseteq (I,H) \wedge \forall (\ell^a, \ell) \in I' : (\ell^{a'}, \ell) \in I) \Rightarrow \beta^{\ell^{a'}}_{me}(P) \sqsubseteq (I,H)$$

Proof. By structural induction on P. ■

The above fact states how, intuitively, the result of $\beta^{\ell^a}_{me}(P)$ can be transformed so as to also include the result of $\beta^{\ell^{a'}}_{me}(P)$; this result is needed to establish the subject reduction property, which we shall prove next:

Theorem 4.17 (Subject Reduction) *Let $P, Q \in \mathsf{Proc}$, $\ell \in \mathsf{Lab}^a$ and $me \in \mathsf{MEnv}$ then $\beta^{\ell}_{me}(P) \sqsubseteq (I,H) \wedge (I,H) \models_{me} P \wedge P \to Q$ implies that $\beta^{\ell}_{me}(Q) \sqsubseteq (I,H) \wedge (I,H) \models_{me} Q$.*

Proof. By structural induction on P.

Case (Red Amb): By assumption we have that

$$n^{\ell^a_n}[P] \to n^{\ell^a_n}[Q]$$

because $P \to Q$. Furthermore, by assumption:

$$\begin{array}{rcl} \beta^{\ell}_{me}(n^{\ell^a_n}[P]) & = & \beta^{\ell^a_n}_{me}(P) \sqcup (\{(\ell, \ell^a_n)\}, \{(\ell^a_n, me(n))\}) \\ & \sqsubseteq & (I, H) \end{array} \tag{4.1}$$

And finally

$$(I, H) \models_{me} n^{\ell^a_n}[P] \iff (I, H) \models_{me} P \tag{4.2}$$

Combining (4.1) and (4.2) we have:

$$\beta^{\ell^a_n}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P \wedge P \to Q$$

which, by induction hypothesis, implies that

$$\beta^{\ell^a_n}_{me}(Q) \sqsubseteq (I, H) \wedge (I, H) \models_{me} Q$$

Using (4.1) with the above, we obtain

$$\begin{array}{rcl} \beta^{\ell}_{me}(n^{\ell^a_n}[Q]) & = & \beta^{\ell^a_n}_{me}(Q) \sqcup (\{(\ell, \ell^a_n)\}, \{(\ell^a_n, me(n))\}) \\ & \sqsubseteq & (I, H) \end{array}$$

and

$$(I, H) \models_{me} Q \iff (I, H) \models_{me} n^{\ell^a_n}[Q]$$

Which proves this case.

Case (Red In): By assumption we have that

$$n^{\ell^a_n}[in^{\ell^t} m.P] \mid Q \mid m^{\ell^a_m}[R] \to m^{\ell^a_m}[n^{\ell^a_n}[P \mid Q] \mid R]$$

And furthermore that

$$\begin{array}{l} \beta^{\ell}_{me}(n^{\ell^a_n}[in^{\ell^t} m.P] \mid Q \mid m^{\ell^a_m}[R]) \\ \quad = \quad \beta^{\ell^a_n}_{me}(in^{\ell^t} m.P \mid Q) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a_n, me(n))\}) \sqcup \\ \qquad\quad\; \beta^{\ell^a_m}_{me}(R) \sqcup (\{(\ell, \ell^a_m)\}, \{(\ell^a_m, me(m))\}) \\ \quad = \quad \beta^{\ell^a_n}_{me}(P) \sqcup \beta^{\ell^a_n}_{me}(Q) \sqcup \beta^{\ell^a_m}_{me}(R) \sqcup \\ \qquad\quad\; (\{(\ell, \ell^a_n), (\ell, \ell^a_m), (\ell^a_n, \ell^t)\}, \{(\ell^a_n, me(n)), (\ell^a_m, me(m))\}) \\ \quad \sqsubseteq \quad (I, H) \end{array} \tag{4.3}$$

And also:

$$\begin{array}{l} (I, H) \models_{me} n^{\ell^a_n}[in^{\ell^t} m.P] \mid Q \mid m^{\ell^a_m}[R] \\ \iff (I, H) \models_{me} in^{\ell^t} m.P \wedge (I, H) \models_{me} Q \wedge (I, H) \models_{me} R \\ \iff (I, H) \models_{me} P \wedge (I, H) \models_{me} Q \wedge (I, H) \models_{me} R \wedge \\ \qquad (\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : \; (\ell^a, \ell^t) \in I \wedge (\ell^{a''}, \ell^a) \in I \wedge \\ \qquad\qquad (\ell^{a''}, \ell^{a'}) \in I \wedge (\ell^{a'}, me(m)) \in H \Rightarrow \\ \qquad\qquad\quad (\ell^{a'}, \ell^a) \in I) \end{array} \tag{4.4}$$

Now from (4.3) we have that $(\ell_n^a, \ell^t) \in I$, $(\ell, \ell_n^a) \in I$, $(\ell, \ell_m^a) \in I$ and $(\ell_m^a, me(m)) \in H$. Thus it follows from (4.4) that $(\ell_m^a, \ell_n^a) \in I$, and furthermore, by (4.3) again, that:

$$\begin{array}{ll} \multicolumn{2}{l}{\beta_{me}^{\ell}(m^{\ell_m^a}[n^{\ell_n^a}[P \mid Q] \mid R])} \\ = & \beta_{me}^{\ell_n^a}(P) \sqcup \beta_{me}^{\ell_n^a}(Q) \sqcup \beta_{me}^{\ell_m^a}(R) \sqcup \\ & (\{(\ell, \ell_m^a), (\ell_m^a, \ell_n^a)\}, \{(\ell_n^a, me(n)), (\ell_m^a, me(m))\}) \\ \sqsubseteq & (I, H) \end{array}$$

Finally, using (4.4), we obtain the following

$$\begin{array}{ll} \multicolumn{2}{l}{(I, H) \models_{me} P \wedge (I, H) \models_{me} Q \wedge (I, H) \models_{me} R} \\ \iff & (I, H) \models_{me} m^{\ell_m^a}[n^{\ell_n^a}[P \mid Q] \mid R] \end{array}$$

Which ends this case.

Case (Red Open): We have, by assumption:

$$open^{\ell^t} n.P \mid n^{\ell_n^a}[Q] \to P \mid Q$$

And

$$\begin{array}{ll} \multicolumn{2}{l}{\beta_{me}^{\ell}(open^{\ell^t} n.P \mid n^{\ell_n^a}[Q])} \\ = & \beta_{me}^{\ell}(open^{\ell^t} n.P) \sqcup \beta_{me}^{\ell}(n^{\ell_n^a}[Q]) \\ = & \beta_{me}^{\ell}(P) \sqcup \beta_{me}^{\ell_n^a}(Q) \sqcup (\{(\ell, \ell^t), (\ell, \ell_n^a)\}, \{(\ell_n^a, me(n))\}) \\ \sqsubseteq & (I, H) \end{array} \tag{4.5}$$

And finally that

$$\begin{array}{ll} \multicolumn{2}{l}{(I, H) \models_{me} open^{\ell^t} n.P \mid n^{\ell_n^a}[Q]} \\ \iff & (I, H) \models_{me} open^{\ell^t} n.P \wedge (I, H) \models_{me} n^{\ell_n^a}[Q] \\ \iff & (I, H) \models_{me} P \wedge (I, H) \models_{me} Q \wedge \\ & (\forall \ell^a, \ell^{a'} \in \mathbf{Lab}^a : \\ & \quad (\ell^a, \ell^t) \in I \wedge (\ell^a, \ell^{a'}) \in I \wedge (\ell^{a'}, me(n)) \in H \Rightarrow \\ & \qquad \left\{(\ell^a, \ell') \,\middle|\, (\ell^{a'}, \ell') \in I\right\} \subseteq I) \end{array} \tag{4.6}$$

By (4.5) we have that $\beta_{me}^{\ell_n^a}(Q) \sqsubseteq (I, H)$ and from (4.6) that

$$\{(\ell, \ell') \mid (\ell_n^a, \ell') \in I\} \subseteq I \iff \forall (\ell_n^a, \ell') \in I : (\ell, \ell') \in I$$

It then follows from Fact 4.16 that $\beta_{me}^{\ell}(Q) \sqsubseteq (I, H)$, whence by (4.5) we have in conclusion that

$$\begin{array}{rcl} \beta_{me}^{\ell}(P \mid Q) & = & \beta_{me}^{\ell}(P) \sqcup \beta_{me}^{\ell}(Q) \\ & \sqsubseteq & (I, H) \end{array}$$

From (4.6) we clearly have

$$(I, H) \models_{me} P \wedge (I, H) \models_{me} Q \iff (I, H) \models_{me} P \mid Q$$

Which proves this case.

The remaining cases are similar or trivial. ■

Apart from establishing the semantic correctness of the analysis, the above theorem will also be used for proving various properties in Chapter 5.

As a consequence of Theorem 4.17 we see that if during an evaluation sequence an ambient, $m^{\ell^a_m}$, enters another ambient, $n^{\ell^a_n}$, then the analysis will reflect this; which, as stated in the introduction of this chapter, is the intention of the analysis. That is if we have the following (simplified) situation:

$$P \rightarrow^* n^{\ell^a_n}[m^{\ell^a_m}[Q] \mid R]$$

and furthermore assume that we have I, H and me such that $\beta^{\ell}_{me}(P) \sqsubseteq (I, H)$ for some $\ell \in \mathbf{Lab}^a$ and $(I, H) \models_{me} P$. Theorem 4.17 then gives us that

$$\beta^{\ell}_{me}(n^{\ell^a_n}[m^{\ell^a_m}[Q] \mid R]) \sqsubseteq (I, H)$$

and

$$(I, H) \models_{me} n^{\ell^a_n}[m^{\ell^a_m}[Q] \mid R]$$

And therefore we have that $(\ell^a_n, \ell^a_m) \in I$, which is how the analysis reflects that the ambient labelled ℓ^a_m at some point (potentially) enters an ambient labelled ℓ^a_m. The same argument holds for individual capabilities.

4.2.4 The existence of solutions

With the semantic correctness of the analysis in place, we now show that every (legal) expression in the Mobile Ambients calculus admits an analysis; furthermore, we show that there always exists a *least* analysis. Both these results are established via a Moore family result, cf. Definition 4.7, showing that the set of analysable expressions form a Moore family. In order to facilitate this proof, we first show three general, but minor, results:

Fact 4.18 *Let L be a complete lattice, such that $\beta \in L$, then $\{x \mid \beta \sqsubseteq x\} \subseteq L$ is a Moore family.*

Proof. Consider $X \subseteq \{x \mid \beta \sqsubseteq x\}$, it then follows that:

$$\begin{aligned} \forall x \in X : \beta \sqsubseteq x \quad &\Rightarrow \quad \beta \sqsubseteq \bigsqcap_{x \in X} x \\ &\Rightarrow \quad (\bigsqcap_{x \in X} x) \in \{x \mid \beta \sqsubseteq x\} \end{aligned}$$

Thus $\{x \mid \beta \sqsubseteq x\}$ is a Moore family. ■

Fact 4.19 *Let L be a complete lattice and $\{x \mid P(x)\}, \{x \mid Q(x)\} \subseteq L$ Moore families. Then $\{x \mid P(x) \wedge Q(x)\}$ is a Moore family.*

Proof. Let $X_P = \{x \mid P(x)\}$, $X_Q = \{x \mid Q(x)\}$ and $X_\wedge = \{x \mid P(x) \wedge Q(x)\}$. Now we have that:

$$\begin{array}{rcl} X_\wedge \subseteq X_P \wedge X_\wedge \subseteq X_Q & \Rightarrow & \forall X \subseteq X_\wedge : (\sqcap X \in X_P) \wedge (\sqcap X \in X_Q) \\ & \Rightarrow & \forall X \subseteq X_\wedge : \sqcap X \in X_P \cap X_Q \\ & \Rightarrow & \forall X \subseteq X_\wedge : \sqcap X \in X_\wedge \end{array}$$

Thus $X_\wedge = \{x \mid P(x) \wedge Q(x)\}$ is a Moore family. ■

The next fact shows that clauses on a certain form are closed under the "greatest lower bound" operator, and hence can be used to construct clauses that have the Moore family property.

Fact 4.20 *Let L be a complete lattice such that $a, b \in L$ and let $x_i \in L$ for $i \in \mathcal{I}$. Then*

$$(\forall i \in \mathcal{I} : a \sqsubseteq x_i \Rightarrow b \sqsubseteq x_i) \Rightarrow (a \sqsubseteq \sqcap x_i \Rightarrow b \sqsubseteq \sqcap x_i)$$

Proof. Assume that

$$\forall i \in \mathcal{I} : a \sqsubseteq x_i \Rightarrow b \sqsubseteq x_i$$

Let $a, b \in L$ and assume furthermore that $a \sqsubseteq \sqcap x_i$. Then we have

$$\begin{array}{rcl} a \sqsubseteq \sqcap x_i & \Rightarrow & \forall i \in \mathcal{I} : a \sqsubseteq x_i \\ & \Rightarrow & \forall i \in \mathcal{I} : b \sqsubseteq x_i \\ & \Rightarrow & b \sqsubseteq \sqcap x_i \end{array}$$

Thus proving the fact. ■

Instantiating the above fact with a specific complete lattice, $L = \mathcal{P}(\mathsf{\textbf{InAmb}}) \times \mathcal{P}(\mathsf{\textbf{HNam}})$ say, gives the following Corollary.

Corollary 4.21 *Consider the complete lattice $(L, \sqsubseteq)$ where $L = \mathcal{P}(\mathsf{\textbf{InAmb}}) \times \mathcal{P}(\mathsf{\textbf{HNam}})$ and $\sqsubseteq$ is defined coordinatewise. Furthermore let $(a, b), (a', b') \in L$ and $(I_i, H_i) \in L$ for $i \in \mathcal{I}$. Then*

$$\begin{array}{l} (\forall i \in \mathcal{I} : (a \subseteq I_i \wedge b \subseteq H_i) \Rightarrow (a' \subseteq I_i \wedge b' \subseteq H_i)) \Rightarrow \\ \qquad ((a \subseteq \cap I_i \wedge b \subseteq \cap H_i) \Rightarrow (a' \subseteq \cap I_i \wedge b' \subseteq \cap H_i)) \end{array}$$

Proof. Trivial instance of Fact 4.20. ■

It should be evident that the Corollary has immediate applications with respect to the analysis:

Lemma 4.22 *Let $P \in$ **Proc** and $me \in$ **MEnv**, then the set of analyses acceptable with respect to the closure property, $\{(I, H) \mid (I, H) \models_{me} P\}$, is a Moore family.*

Proof. By structural induction on P. Given $me \in \textbf{MEnv}$ and assuming that

$$\forall i \in \mathcal{I} : (I_i, H_i) \models_{me} P$$

we must show that $\sqcap_i(I_i, H_i) \models_{me} P$. In the following we let $I = \cap_i I_i$, $H = \cap_i H_i$ and $(I, H) = \sqcap_i(I_i, H_i)$.

Case ($(\nu_\mu n)P$): We have

$$\forall i \in \mathcal{I} : (I_i, H_i) \models_{me} (\nu_\mu n)P \iff \forall i \in \mathcal{I} : (I_i, H_i) \models_{me[n \mapsto \mu]} P$$

Now by induction hypothesis it follows that

$$(I, H) \models_{me[n \mapsto \mu]} P \iff (I, H) \models_{me} (\nu_\mu n)P$$

Which proves this case.

Case ($in^{\ell^t} n.P$): By assumption

$$\begin{array}{l} \forall i \in \mathcal{I} : (I_i, H_i) \models_{me} in^{\ell^t} n.P \\ \quad \iff \quad \forall i \in \mathcal{I} : ((I_i, H_i) \models_{me} P \wedge \\ \qquad\qquad\qquad (\forall \ell^a, \ell^{a'}, \ell^{a''} \in \textbf{Lab}^a : \\ \qquad\qquad\qquad\quad (\ell^a, \ell^t) \in I_i \wedge (\ell^{a''}, \ell^a) \in I_i \wedge (\ell^{a''}, \ell^{a'}) \in I_i \wedge \\ \qquad\qquad\qquad\quad (\ell^{a'}, me(n)) \in H_i \Rightarrow (\ell^{a'}, \ell^a) \in I_i)) \end{array}$$

By induction hypothesis we obtain $(I, H) \models_{me} P$. Consider now the clause:

$$\begin{array}{r} \forall i \in \mathcal{I} : \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textbf{Lab}^a : (\ell^a, \ell^t) \in I_i \wedge (\ell^{a''}, \ell^a) \in I_i \wedge (\ell^{a''}, \ell^{a'}) \in I_i \wedge \\ (\ell^{a'}, me(n)) \in H_i \Rightarrow (\ell^{a'}, \ell^a) \in I_i \end{array}$$

equivalent to the clause:

$$\begin{array}{l} \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textbf{Lab}^a : \forall i \in \mathcal{I} : \\ \quad \Big\{(\ell^a, \ell^t), (\ell^{a''}, \ell^a), (\ell^{a''}, \ell^{a'})\Big\} \subseteq I_i \wedge \Big\{(\ell^{a'}, me(n))\Big\} \subseteq H_i \Rightarrow \\ \qquad \Big\{(\ell^{a'}, \ell^a)\Big\} \subseteq I_i \wedge \emptyset \subseteq H_i \end{array}$$

which is clearly of the form of Corollary 4.21. Using that Corollary we have:

$$\begin{array}{r} \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textbf{Lab}^a : (\ell^a, \ell^t) \in I \wedge (\ell^{a''}, \ell^a) \in I \wedge (\ell^{a''}, \ell^{a'}) \in I \wedge \\ (\ell^{a'}, me(n)) \in H \Rightarrow (\ell^{a'}, \ell^a) \in I \end{array}$$

Thus, in conclusion, we have

$$(I, H) \models_{me} in^{\ell^t} n.P$$

Which proves this case.

The remaining cases are similar or trivial. ■

Now we are in a position to prove the following theorem:

Theorem 4.23 *Let $P \in \textsf{Proc}$ and $\ell \in \textsf{Lab}^a$ and let $me \in \textsf{MEnv}$, then the set $\left\{(I,H) \,\middle|\, (I,H) \models_{me} P \wedge \beta^{\ell}_{me}(P) \sqsubseteq (I,H)\right\}$ is a Moore family.*

Proof. From Fact 4.22 we have that the set $\{(I,H) \mid (I,H) \models_{me} P\}$ is a Moore family, and from Fact 4.18 it follows that the set $\left\{(I,H) \,\middle|\, \beta^{\ell}_{me}(P) \sqsubseteq (I,H)\right\}$ is a Moore family. Finally the theorem follows from Fact 4.19. ■

Now let $\mathcal{A} = \left\{(I,H) \,\middle|\, (I,H) \models_{me} P \wedge \beta^{\ell}_{me}(P) \sqsubseteq (I,H)\right\}$ be the set of admissible analyses of the process $P \in \textsf{Proc}$. It then follows from Theorem 4.23 that every process, P, admits an analysis. To see this note that $\sqcap\emptyset$ is an element of $\mathcal{A}$. Furthermore, every process has a *least* analysis, which follows from the fact that $\sqcap\mathcal{A}$ must be an element of $\mathcal{A}$, and clearly $\sqcap\mathcal{A}$ is the least element of $\mathcal{A}$ and hence a least analysis for P.

Remark 4.24 The approach taken in this section was slightly more abstract and general than was strictly necessary in order to establish the main results. This was done to emphasise that it is possible to construct analyses that *by definition* have the Moore family property, i.e. analyses defined solely by simple inclusions (Fact 4.18), logical conjunctions (Fact 4.19) and (a simple form of) implications (Fact 4.20) are guaranteed to have the Moore family property. ◇

Chapter 5

Computer Security in Mobile Ambients

In this chapter we study how the development from the previous chapters may be applied to the field of computer security. Arguably computer security has become more important than ever with the seemingly endless growth of large scale computer networks. In particular the use of mobile code, or mobile agents, on these networks represents a fundamental change in how computer security is perceived: assumptions previously basic to computer security does no longer hold in an environment using or allowing for mobile code. For instance, there is no longer a trivial mapping between processes and users/owners. A given process may be part of a downloaded agent, executing on behalf of some remote, and possibly unknown, user that may or may not have any relation to the machine running the process (or the owner(s) of the machine). This makes it harder, if not impossible, to base computer security (solely) on user identity. Further discussion of these and other points can be found in [Che98]; and also [Lan81] for an overview of general computer security aspects.

In showing how the analysis from Chapter 4 can be used to guarantee various security properties in a Mobile Ambients model of a computer system, we shall take an approach similar to [BDNN98, BDNN99]. In this approach, first a dynamic notion is identified. This dynamic notion should capture the (desired) operational aspects of the security property in question. Next a corresponding static notion is defined, keeping in mind that the notion should be statically checkable, i.e. it should be decidable to check if a process has this property; and the static property should be a sufficient condition for the dynamic property, in the sense that processes/systems that has the static property will also have the dynamic property. In the sequel, as in [BDNN98, BDNN99], the static property will be defined in terms of a control flow analysis, specified by a flow logic, cf. Chapter 4, thus guaranteeing that the static notion is statically checkable.

Other recent work, such as [Aba97, HR98], has been using type systems to statically check and ensure that security properties hold for a given process/program. In [Aba97] the spi calculus, an extension of the π-calculus with cryptographic primitives, is used to model cryptographic protocols and a type system on the spi calculus is used to guarantee certain security properties of the protocols. The goal of [HR98] is not, as such, to validate the soundness of cryptographic protocols modelled in a process calculus. Rather, it is an attempt at introducing an explicit notion of *security information* into a typed λ-calculus, called the Secure Lambda Calculus (the SLam calculus). This is done by maintaining not only type information but also security information for objects in the calculus. To ensure that no security violations can occur, the semantics of the SLam calculus incorporates several dynamic (runtime) checks. In order to get rid of these dynamic checks a type system is constructed that statically guarantees that no security violations can occur. The approach we shall take in the following, as described above, is obviously very close in spirit to the one taken in [HR98].

One major advantage of using a flow logic rather than a type system, as in [Aba97, HR98] described above, is that it is always possible to construct an algorithm to check for the static notion: for a given (finite) process, simply generate and check all the finitely many possible solutions to the control flow analysis. Moreover, by establishing a Moore family result for the analysis, the existence of least solutions is guaranteed and can be calculated by computing the "intersection", in some sense, of all the acceptable analyses. In contrast, it is not obvious whether or not there exists principal types for the type systems of [Aba97] and [HR98].

Another approach to formal verification of security properties, that we shall only briefly comment upon here, is that of [AG98] where barbed bisimulation and testing equivalence is used to verify the correctness of cryptographic protocols modelled in the spi calculus. Essentially the idea is to check if the model of the protocol to be validated is barbed bisimilar or testing equivalent to its specification. While this approach seems quite powerful and flexible, it is, as with type systems, not immediately obvious how or if the process of validation can be automated.

In the following sections we define some abstract notions of security, based on rather simple models/setups. This makes them suitable as "building blocks" for more complex notions of security.

5.1 Preliminaries

In order to facilitate the presentation of formal security properties in the following sections, we introduce some useful notation and technical conveniences in this section.

As explained in Chapter 4, if (I, H) is an acceptable analysis then $(\ell^a, \ell) \in I$ indicates that an ambient or a capability with the label ℓ may at some point during evaluation end up inside an ambient labelled ℓ^a. For some of the notions in the following sections the "inverse" of an analysis is of interest, i.e. given a capability or an ambient labelled ℓ what ambients might it end up inside. This is formalised in the next definition:

Definition 5.1 *Let* $I \subseteq \textsf{Lab}^a \times \textsf{Lab}$*, and define*

$$I^{-1} = \{(\ell', \ell) \,|\, (\ell, \ell') \in I\}$$

While an analysis, I, is concerned with the immediate constituents of a given ambient, it may be used "transitively" to find out about ambients or capabilities at deeper levels. Furthermore, using an analysis, I, in conjunction with the derived I^{-1}, the set of potential siblings (both capabilities and ambients) can be calculated:

Definition 5.2 *Let* $I \subseteq \textsf{Lab}^a \times \textsf{Lab}$*, and define*

$$II = \{(\ell, \ell'') \,|\, \exists \ell' : (\ell, \ell') \in I \wedge (\ell', \ell'') \in I\}$$

and

$$II^{-1} = \left\{(\ell, \ell'') \,\middle|\, \exists \ell' : (\ell, \ell') \in I^{-1} \wedge (\ell', \ell'') \in I\right\}$$

Thus if $(\ell^a, \ell) \in II$ then the ambient or capability labelled ℓ may at some point be inside an ambient that itself may reside inside an ambient labelled ℓ^a. Similarly, if $(\ell, \ell') \in II^{-1}$ then the capability or ambient represented by ℓ may at some point end up as a sibling to a capability or an ambient labelled ℓ'. Furthermore, we shall write $(\ell^a, \ell) \in I^*$ if there exists $\ell^a_1, \ldots, \ell^a_n$ such that $\ell^a = \ell^a_1$, $\ell = \ell^a_n$ and $(\ell^a_i, \ell^a_{i+1}) \in I$ for $1 \leq i \leq n$.

Rather than dealing explicitly with α-equivalence in the following sections, we introduce the notion of *stable capabilities*. The set of stable capabilities, **CAPA**, is defined as follows:

$$\textsf{CAPA} = \left\{\textsf{in}^{\ell^t} \mu, \textsf{out}^{\ell^t} \mu, \textsf{open}^{\ell^t} \mu \,\middle|\, \ell^t \in \textsf{Lab}^t \wedge \mu \in \textsf{Mar}\right\}$$

The intuition being that markers are used instead of names, because markers are, by definition, invariant under α-equivalence. The set of stable capabilities in a given process is defined as follows:

Definition 5.3 (Occurring Stable Capabilities) *For* $me \in \textsf{MEnv}$ *and* $P \in \textsf{Proc}$ *let* $\mathrm{CAPA}_{me}(P) \subseteq \textsf{CAPA}$ *be the set of* occurring stable capabilities *in* P*,* *defined as follows:*

$$\begin{array}{rcl}
\mathrm{CAPA}_{me}(\mathbf{0}) & = & \emptyset \\
\mathrm{CAPA}_{me}(!P) & = & \mathrm{CAPA}_{me}(P) \\
\mathrm{CAPA}_{me}(P \mid Q) & = & \mathrm{CAPA}_{me}(P) \cup \mathrm{CAPA}_{me}(P) \\
\mathrm{CAPA}_{me}((\nu_\mu n)P) & = & \mathrm{CAPA}_{me[n \mapsto \mu]}(P) \\
\mathrm{CAPA}_{me}(n^{\ell^a_n}[P]) & = & \mathrm{CAPA}_{me}(P) \\
\mathrm{CAPA}_{me}(in^{\ell^t} n.P) & = & \left\{\mathsf{in}^{\ell^t} \mu\right\} \cup \mathrm{CAPA}_{me}(P) \\
 & & \quad \textit{if } me(n) = \mu \\
\mathrm{CAPA}_{me}(out^{\ell^t} n.P) & = & \left\{\mathsf{out}^{\ell^t} \mu\right\} \cup \mathrm{CAPA}_{me}(P) \\
 & & \quad \textit{if } me(n) = \mu \\
\mathrm{CAPA}_{me}(open^{\ell^t} n.P) & = & \left\{\mathsf{open}^{\ell^t} \mu\right\} \cup \mathrm{CAPA}_{me}(P) \\
 & & \quad \textit{if } me(n) = \mu
\end{array}$$

Note that a marker environment is needed in order to define the scope (marker) of the free names in a given process. The marker used for bound occurrences of a name is the marker used in the corresponding ν-operator.

A further technical convenience is adopted in this chapter: it is assumed that all the names in a given process, $P \in \mathbf{Proc}$, is annotated with a marker, e.g. $n_\mu^{\ell^a_n}$. The marker represents the scope of that name, and thus bound names are annotated with the marker for the corresponding ν-operator. In the case of free names we have to take care that whenever a marker environment is needed, the scopes assigned by the marker environment must not be in conflict with the annotated scopes. Hence we shall say that a marker environment me that assigns to the free names of P the corresponding annotation is said to be *compatible* with P. Formally we define:

Definition 5.4 *A marker environment, $me \in$ **MEnv**, is said to be* compatible *with $P \in$ **Proc** if:*

$$\forall n_\mu^{\ell^a_n} \in fn(P) : me(n) = \mu$$

Note that with this definition we tacitly assume that all bound names are correctly annotated with the marker of the corresponding binding operator.

The reason for this technical machinery will become evident in the sections to follow, in particular the definitions and proofs of Sections 5.2 and 5.4 benefit from these notions. It should be noted that the notion of base step, cf. Definition 3.6, is not affected by the annotation.

It would have been more elegant, perhaps, to incorporate this "distributed scope" into the semantics, e.g. by using reduction contexts or similar mechanisms, as was done in [HJNN99]. It is, however, only needed in this chapter, and would seem to complicate matters unnecessarily in the previous and following chapters.

5.2 Restricting the use of *open*-capabilities

Dynamic notion. In this section we will briefly discuss how the presence of *open*-capabilities can be handled; in particular how it is possible to control exactly what processes/threads are allowed to open one of a predetermined set of ambients. This notion can be used in connection with the problem of "malicious hosts", cf. [Che98, Section 3.5], in the sense that it may be possible to test a host before sending it a mobile agent, in order to ensure that the host is not "malicious". In this context, a "malicious host" is a host that destroys (opens) a mobile agent (ambient). This can be modelled by the following simple expression in the ambient calculus:

$$a_{\mu}^{\ell_a^a}[in^{\ell^t} h_{\mu'}.P] \mid h_{\mu'}^{\ell_h^a}[Q]$$

where $a_{\mu}^{\ell_a^a}$ represents an agent accessing a host, represented by $h_{\mu'}^{\ell_h^a}$. Here we are interested in guaranteeing that the host does not open $a_{\mu}^{\ell_a^a}$, i.e. that Q does not contain capabilities that can be used to open $a_{\mu}^{\ell_a^a}$. Note that this notion of "maliciousness" is rather primitive in that it does not restrict or check for the host's ability to read and/or copy the contents of the agent. However, the notions defined in Sections 5.3 and 5.4 may be used for this purpose.

In order to formalise the above, we generalise the situation and introduce the set $\mathcal{R}_{open}$, representing the set of ambients that are allowed to be opened only by a capability from a predetermined set of stable *open*-capabilities, represented by $\mathcal{A}_{open}$. We first define the desired dynamic property:

Definition 5.5 (Restricted open) *Let* $P \in \textsf{Proc}$*,* $\mathcal{A}_{open} \subseteq \textsf{CAPA}$ *and let* $\mathcal{R}_{open} \subseteq \textsf{Lab}^a \times \textsf{Mar}$*. Process* P *performs only* restricted open wrt. $(\mathcal{A}_{open}, \mathcal{R}_{open})$ *iff* $\forall Q, R \in \textsf{Proc}$ *such that*

$$P \rightarrow^* Q \rightarrow R$$

and the base step of $Q \rightarrow R$ *is in the form:*

$$open^{\ell^t} n_{\mu}.S \mid n_{\mu}^{\ell_n^a}[T] \rightarrow S \mid T$$

then $(\ell_n^a, \mu) \in \mathcal{R}_{open} \Rightarrow \mathsf{open}^{\ell^t}\mu \in \mathcal{A}_{open}$.

Intuitively the definition states that only *open*-capabilities described by $\mathcal{A}_{open}$ are allowed to be used on ambients from $\mathcal{R}_{open}$.

A point worth noting about the definition is the use of markers and ambient labels to specify a particular ambient, rather than only its name and/or label. This is mostly a technical convenience facilitating the proof of Proposition 5.8 in the presence of α-equivalence; under the assumption of uniquely labelled capabilities and ambients this machinery would not be necessary.

Static notion. It can be proved that in general it is undecidable to determine whether or not an expression, given $\mathcal{A}_{open}$ and $\mathcal{R}_{open}$, has the property of Definition 5.5. Instead we turn towards finding a statically checkable property that will ensure the above dynamic property.

For this we shall need the set II^{-1}; cf. Section 5.1. Intuitively if $(\ell, \ell') \in II^{-1}$ then the ambient or capability labelled ℓ may at some point in a derivation end up as a "sibling" to a capability or ambient with the label ℓ'. Thus the set of *open*-capabilities in P that may potentially open a given ambient, $n_\mu^{\ell_n^a}$ say, also in P, is the following:

$$\left\{\mathsf{open}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \,\middle|\, (\ell_n^a, \ell^t) \in II^{-1}\right\} \tag{5.1}$$

for some $me \in \mathbf{MEnv}$ that is compatible with P. Now, given an ambient, $n_\mu^{\ell_n^a}[Q]$, occurring in $P \in \mathbf{Proc}$, a marker environment, $me \in \mathbf{MEnv}$ compatible with P, $\ell \in \mathbf{Lab}^a$, and given a set, $\mathcal{A}_{open} \subseteq \mathbf{CAPA}$, of labels of threads that are allowed to open ambient $n_\mu^{\ell_n^a}$, it is easy to statically check if the process, P, respects this, all that is needed is for the following to hold: there exists I, H such that

$$\beta_{me}^{\ell}(P) \sqsubseteq (I, H) \ \wedge \ (I, H) \models_{me} P$$

and the set defined in (5.1) must be a subset of $\mathcal{A}_{open}$. If these conditions are met, only capabilities represented in $\mathcal{A}_{open}$ will ever be in a position to open $n^{\ell_n^a}$. This is formalised in the following static property:

Definition 5.6 (Controlled open) *Let* $P \in \mathbf{Proc}$, $\mathcal{A}_{open} \subseteq \mathbf{CAPA}$, $\mathcal{R}_{open} \subseteq \mathbf{Lab}^a \times \mathbf{Mar}$ *and let* $\ell \in \mathbf{Lab}^a$ *and* $me \in \mathbf{MEnv}$. *Now P is said to* control *open* wrt. $(\mathcal{A}_{open}, \mathcal{R}_{open}); \ell, me$ *iff me is compatible with P and there exists I and H such that*

$$\beta_{me}^{\ell}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\forall (\ell^a, \mu) \in \mathcal{R}_{open} : \left\{\mathsf{open}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \,\middle|\, (\ell^a, \ell^t) \in II^{-1}\right\} \subseteq \mathcal{A}_{open}$$

The $\ell \in \mathbf{Lab}^a$ in the above definition is included only for technical reasons.

Intuitively Definition 5.6 checks that every *open*-capability that could potentially open the given ambient is allowed to do so. Thus it is a check against potentially dangerous *capabilities*; in Section 5.3 another approach is taken: rather than checking for dangerous capabilities via the $\mathrm{CAPA}_{me}(P)$ function, the analysis is used directly to find out where ambients may end up, and thus check if a malicious ambient can potentially end up inside another ambient.

Semantic Soundness. In order to show that 'controlled open' is useful for checking if a process has the property of 'restricted open', it remains to be proved that 'controlled open' is a sufficient condition for 'restricted open'. We shall need the following fact to prove the semantic soundness:

Fact 5.7 *Let $P \in$ **Proc** and let I, H, $\ell \in \mathsf{Lab}^a$ and $me \in$ **MEnv** be given such that $\beta^{\ell}_{me}(P) \sqsubseteq (I, H)$ and $(I, H) \models_{me} P$, then for all subprocesses, Q, of P we have that there exists me' and ℓ' such that $\beta^{\ell'}_{me'}(Q) \sqsubseteq (I, H)$ and $(I, H) \models_{me'} Q$.*

Proof. By structural induction on P. ■

We can now prove the semantic soundness of the definitions:

Proposition 5.8 *Let $P \in$ **Proc**, $\ell \in \mathsf{Lab}^a$, $me \in$ **MEnv**, $\mathcal{A}_{open} \subseteq$ **CAPA** and let $\mathcal{R}_{open} \subseteq \mathsf{Lab}^a \times \mathsf{Mar}$. If P controls open wrt. $(\mathcal{A}_{open}, \mathcal{R}_{open}); \ell, me$ then P performs only restricted open wrt. $(\mathcal{A}_{open}, \mathcal{R}_{open})$.*

Proof. We have the following situation:

$$P \rightarrow^* Q \rightarrow R$$

With the basestep of $Q \rightarrow R$ on the form below

$$open^{\ell^t} n_\mu.S \mid n_\mu^{\ell^a_n}[T] \rightarrow S \mid T \tag{5.2}$$

Since P controls open wrt. $(\mathcal{A}_{open}, \mathcal{R}_{open}); \ell, me$ we have that there exists I and H such that

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

And from Theorem 4.17 (subject reduction) it follows that:

$$\beta^{\ell}_{me}(Q) \sqsubseteq (I, H) \wedge (I, H) \models_{me} Q$$

Now using Fact 5.7 with (5.2) we have that, for some $me' \in$ **MEnv** and some $\ell' \in \mathsf{Lab}^a$:

$$\beta^{\ell'}_{me'}(open^{\ell^t} n_\mu.S \mid n_\mu^{\ell^a_n}[T]) \sqsubseteq (I, H)$$

In particular: $(\ell^a_n, \ell^t) \in II^{-1}$. Moreover, since $\mathsf{open}^{\ell^t}\,\mu \in \mathrm{CAPA}_{me}(P)$ it follows from Definition 5.6 that $\mathsf{open}^{\ell^t}\,\mu \in \mathcal{A}_{open}$. This ends the proof. ■

Proposition 5.8 gives us a simple way to check if a given expression performs only restricted open (with respect to some $(\mathcal{A}_{open}, \mathcal{R}_{open})$): First we analyse the expression, i.e. generate constraints and solve them as described in Chapter 6; this takes care of the first condition in Definition 5.6. Next $\mathrm{CAPA}_{me}(P)$ is calculated and the solved for I-component is used to check the last condition in Definition 5.6. We now give a few examples of this:

Example 5.9 (Example 3.1 revisited) Consider again the expression from Example 3.1, rewritten in the style of this chapter:

$$(\nu_\mu m)(n_{\mu'}^{\ell^a_n}[in^{\ell^t_1} m_\mu] \mid m_\mu^{\ell^a_m}[open^{\ell^t_2} n_{\mu'}])$$

As discussed previously, this has no effect on the semantics of the expression. For convenience we name the above expression P. In Example 4.1 we saw that

$$\beta^{\ell}_{me}(\mathsf{P}) \sqsubseteq (I, H) \wedge (I, H) \models_{me} \mathsf{P}$$

for some $\ell \in \mathbf{Lab}^a$ and where I, H and me are defined as follows:

$$\begin{array}{rcl} I & = & \{(\ell, \ell^a_n), (\ell, \ell^a_m), (\ell^a_n, \ell^t_1), (\ell^a_m, \ell^a_n), (\ell^a_m, \ell^a_m), (\ell^a_m, \ell^t_1), (\ell^a_m, \ell^a_2)\} \\ H & = & \{(\ell^a_n, \mu'), (\ell^a_m, \mu)\} \\ me & = & [n \mapsto \mu'] \end{array}$$

Having defined an me (that is clearly compatible with P we can now calculate $\mathrm{CAPA}_{me}(\mathsf{P})$:

$$\mathrm{CAPA}_{me}(\mathsf{P}) = \left\{\mathsf{in}^{\ell^t_1}\mu, \mathsf{open}^{\ell^t_2}\mu'\right\}$$

Now let $\mathcal{R}_{open} = \{(\ell^a_n, \mu')\}$ and $\mathcal{A}_{open} = \left\{\mathsf{open}^{\ell^t_2}\mu'\right\}$. Given the above analysis we calculate:

$$\left\{\mathsf{open}^{\ell^t}\mu' \in \mathrm{CAPA}_{me}(\mathsf{P}) \,\middle|\, (\ell^a_n, \ell^t) \in II^{-1}\right\} = \left\{\mathsf{open}^{\ell^t_2}\mu'\right\}$$

And since $\mathcal{R}_{open} = \{(\ell^a_n, \mu')\}$ we have that

$$\forall(\ell^a, \mu) \in \mathcal{R}_{open} : \left\{\mathsf{open}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(\mathsf{P}) \,\middle|\, (\ell^a, \ell^t) \in II^{-1}\right\} \subseteq \mathcal{A}_{open}$$

In other words: P controls open with respect to $(\mathcal{A}_{open}, \mathcal{R}_{open})$, and by Proposition 5.8 process P performs only restricted open with respect to $(\mathcal{A}_{open}, \mathcal{R}_{open})$.

In this very simple example, it is easily seen that had we not chosen to let $\mathsf{open}^{\ell^t_2}\mu' \in \mathcal{A}_{open}$ (maintaining the same $\mathcal{R}_{open}$) the expression would *not* have the above properties. $\diamond$

Example 5.10 We now revisit the expression from the introduction to this section:

$$a^{\ell^a_a}_{\mu}[in^{\ell^t} h_{\mu'}.P] \mid h^{\ell^a_h}_{\mu'}[Q]$$

Now, if we let $\mathcal{R}_{open} = \{(\ell^a_a, \mu)\}$ and, for simplicity, $\mathcal{A}_{open} = \emptyset$; then if the process controls open with respect to $(\mathcal{A}_{open}, \mathcal{R}_{open})$ the host can/will not open the agent. Thus it is conceivable to check if a given host has "malicious" intentions, in the sense of destroying an agent, towards a given agent. Moreover, it may be possible for the agent *itself* to perform this check before entering a given host. In Section 5.2.1 this idea is developed further. $\diamond$

Remark 5.11 In [CGG99] a type system is developed, in part, to statically check that none of the so-called *locked* ambients can be opened during evaluation. The type system can be seen as (another) static check for (a limited version of) the 'restricted open' property, where $\mathcal{R}_{open}$ is chosen to be the set of locked ambients and $\mathcal{A}_{open} = \emptyset$. The type system defined in [CGG99] amounts to a syntactic check for occurrences of *open*-capabilities with the right name(s). In contrast, the approach of checking for 'controlled open' allows for syntactic occurrences of *open*-capabilities in the process, even using the names of restricted ambients, provided that none of these *open*-capabilities can ever be used to open any of the restricted ambients. This is possible due to the added precision of using the control flow analysis for tracking where *open*-capabilities may end up during evaluation. ◇

5.2.1 Applying restricted/controlled open

The notions of restricted and controlled open are somewhat "low-level": it would be very cumbersome to formulate and enforce security policies in terms of these notions. In this section we will discuss, informally, how higher level security primitives, concerning *open*-capabilities, may be described in terms of restricted/ controlled open. In particular how to apply restricted/controlled open to a situation where a given set of ambients (agents) may only be opened inside a predetermined set of other ambients (hosts). This abstraction allows us to do away with the need to explicitly specify the particular *open*-capabilities that may be used; instead they are defined indirectly. Intuitively this will allow us to state which agents are allowed access rather how they do it.

In order to illustrate the notions to be defined, consider again the following expression, cf. Example 5.10:

$$a_{\mu}^{\ell_a^a}[in^{\ell^t} h_{\mu'}.P] \mid h_{\mu'}^{\ell_h^a}[Q] \tag{5.3}$$

As noted above we would like to ensure that the agent, $a_{\mu}^{\ell_a^a}$, is only opened *inside* the host, $h_{\mu'}^{\ell_h^a}$ and nowhere else. If we let

$$\mathcal{R}_{open} = \{(\ell_a^a, \mu)\}$$

and

$$\mathcal{A}_{open} = \left\{ \mathsf{open}^{\ell^t} \mu \,\middle|\, (\ell_h^a, \ell^t) \in I \wedge \{\ell \mid (\ell^t, \ell) \in I^{-1}\} = \{\ell_h^a\} \right\}$$

and check if the expression of (5.3) has the property of controlled open with respect to $(\mathcal{A}_{open}, \mathcal{R}_{open})$, it then follows from Proposition 5.8 that only the *open*-capabilities specified by $\mathcal{A}_{open}$ can ever be in a position to open the agent, $a_{\mu}^{\ell_a^a}$. Given the particular definition of $\mathcal{A}_{open}$ it is evident that the *open*-capabilities defined by $\mathcal{A}_{open}$ are those that are inside the host from the outset and never leaves. The above definition of $\mathcal{A}_{open}$ furthermore requires that the *open*-capabilities are immediate constituents of the host, i.e. "live" at the top level of the host; this somewhat restrictive definition may of course be relaxed according to the situation at hand. One obvious alternative would be to allow the relevant *open*-capabilities to reside in any ambient that *never* leaves the host. However, since it is not entirely trivial to capture the notion of an ambient never leaving another ambient, we defer the discussion of this issue until Section 5.4, where these problems are studied in full.

5.3 Allowed interaction

An important security question to consider when allowing mobile agents, such as Java applets, access to a private network/system, is how to ensure that the agent does not "steal" or destroy critical data. In other words: how does a given agent interact with its environment, and how can one restrict or control the interaction. In this section we describe a possible solution to the problem for systems modelled in the ambient calculus: a dynamic notion is defined, capturing the notion that only a predetermined set of agents are allowed to enter certain, also predetermined, agents or domains. This notion ensures that areas or domains containing critical or sensitive data, are only entered by agents that are specifically allowed to interact with the given domain and hence it can be used to control and/or restrict how an agent interacts with the private system.

The problems described above are of interest not only in conjunction with mobile agents and applets, they apply also to the situation of installing new software. A common problem when installing new software is to decide whether the new software will interfere in an unfortunate or unforeseen manner with already installed software, e.g. either through unlucky coincidence or because the new software is virus infected. This may, at best, result in slight inconveniences for the user(s) and, at worst, in breach of security, loss of data or general systems failure. The solution proposed above, however, can be adapted to this situation: before installing new software it is scanned for potential risks and rejected if it *might* cause problems.

5.3.1 The dynamic notion of allowed interaction

As indicated by the previous discussion, we shall define a dynamic notion that will ensure that only a predetermined set of agents are allowed to enter a given set of agents/domains. In order to do this we must first model the situation in the mobile ambients calculus. The discussion leads us to naturally focus our attention on the *in*-capability and how to control its usage.

We proceed by an informal discussion of how to model the situation at hand, followed by a formalisation of the points discussed. Note that in the informal discussion we dispense with labels and markers and other similar machinery added for the benefit of the analysis.

Let the ambient c represent the private system, e.g. an Internet server, and let P and A represent the private areas, e.g. system critical data, and the mobile agent respectively. Alternatively, A can also be thought of as some piece of "new software" to be installed and P as the previously installed software, not to be interfered with by the new software. Then the scenarios discussed may be modelled by the following process

$$c[P \mid A]$$

Having made this simple model of the basic setup, we turn towards defining the concept of one process or program engaging in allowed interaction with another.

Adopting the view that the process P, in the model described above, must be protected from process A, i.e. process P must be allowed to run without interference from A; we say that process A engages only in allowed interaction with process P if and only if every ambient entering any ambient in P, must itself belong to P. In other words: if $P \mid A \to^* Q \to R$ and the base step of $Q \to R$ (cf. Definition 3.6) is on the form

$$n[in\ w.S \mid T] \mid w[U] \to w[n[S \mid T] \mid U]$$

and if w is an ambient in P then so is n. This definition is rather restrictive, in that it prohibits P from receiving direct communication from the "outside". It is, however, possible for P to send out "probes" that can test, e.g. for the existence of other ambients and then return to P, thus establishing a primitive form of communication with its environment. Now we have the situation that if P and A satisfy the above condition, the "new" process, A, cannot possibly interfere with P (or rather the *internal* workings of P). It does not, however, preclude A from using an *open*-capability to destroy P. This is a separate issue dealt with in Section 5.2.

In general we may be interested in allowing certain processes access to P, i.e., allow certain ambients that are not already part of P to enter ambients in P. Furthermore it is often desirable to distinguish between the processes that allow only restricted access and the processes that are allowed to access such processes. This leads to the following definition of the *dynamic* notion of allowed interaction, where $\mathcal{R}_{in} \subseteq \mathsf{Lab}^a$ represents the ambients that only explicitly specified ambients are allowed to enter. These special ambients are represented by the set $\mathcal{A}_{in} \subseteq \mathsf{Lab}^a$. Ambients represented in $\mathcal{R}_{in}$ are called *restricted ambients* and ambients represented in $\mathcal{A}_{in}$ are called *allowed ambients*.

Definition 5.12 (Allowed interaction) *Let $P \in \mathsf{Proc}$, $\mathcal{A}_{in}, \mathcal{R}_{in} \subseteq \mathsf{Lab}^a$. Process P is said to engage only in* allowed interaction *with respect to $(\mathcal{A}_{in}, \mathcal{R}_{in})$ iff $\forall Q, R \in \mathsf{Proc}$ such that*

$$P \to^* Q \to R$$

if the base step of $Q \to R$ is on the form:

$$n_{\mu}^{\ell_n^a}[in^{\ell^t} w_{\mu'}.S \mid T] \mid w_{\mu'}^{\ell_w^a}[U] \to w_{\mu'}^{\ell_w^a}[n_{\mu}^{\ell_n^a}[S \mid T] \mid U]$$

then $\ell_w^a \in \mathcal{R}_{in} \Rightarrow \ell_n^a \in \mathcal{A}_{in}$

It is worth noting that 'allowed interaction' merely prevents certain ambients from using *in*-capabilities to enter restricted ambients. In general, however, it does not prevent an ambient, called a "rogue" ambient, from entering one of the allowed ambients, thus possibly being "carried" inside a restricted ambient by (inside) an allowed ambient. From there the "rogue" ambient can use an *out*-capability and end up inside a restricted ambient. If this behaviour is not desirable, it can be avoided by choosing $\mathcal{A}_{in} \subseteq \mathcal{R}_{in}$. Following Definition 5.13, these points will be discussed further.

Thus we now have the dynamic notion of allowed interaction, i.e. we have identified an operational property that we would like our system to have. This leads to the subject of the next section: how to statically infer that a given process has the above property, which corresponds to scanning the program in the above scenario.

5.3.2 The static notion of respect

We now turn towards defining a static notion corresponding to the dynamic notion of 'allowed interaction'. The intuition the following definition is intended to capture, mirroring that of 'allowed interaction', is that a set of distinguished ambients is chosen, represented by $\mathcal{R}_{in}$ below, and the only ambients that may enter such an ambient, is an ambient that is explicitly allowed to do so, represented by the set $\mathcal{A}_{in}$; what enters other ambients is of no consequence, neither is it of any consequence where capabilities may end up. In a subsequent section we show that this is sufficient to guarantee that the above dynamic notion of 'allowed interaction' also holds.

Recalling that the I component of an analysis is a set of ordered pairs, (ℓ, ℓ'), indicating that an ambient or a capability labelled ℓ' may potentially end up inside an ambient with label ℓ, we arrive at the following definition of the static property:

Definition 5.13 (Respect) *Let* $P \in \textsf{Proc}$, $\ell \in \textsf{Lab}^a$ *and* $me \in \textsf{MEnv}$ *and let* $\mathcal{A}_{in}, \mathcal{R}_{in} \subseteq \textsf{Lab}^a$. *The process* P respects $(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ *iff* me *is compatible with* P *and there exists* I *and* H *such that*

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\{\ell' \in \textsf{Lab}^a \mid (\ell, \ell') \in I \wedge \ell \in \mathcal{R}_{in}\} \subseteq \mathcal{A}_{in}$$

In contrast to Definition 5.6 the notion of 'respect' *does not* restrict or control the individual *in*-capabilities; for this reason it is not strictly necessary that me is compatible with P and is only required for consistency. Instead a more direct approach is taken, which is possible because, as indicated by the remark immediately preceding the above definition, this is exactly what the analysis is intended to do: calculate which ambients may end up in which other ambients, cf. Chapter 4.

As noted following its definition, the 'allowed interaction' property in general does not prevent "rogue" ambients from being carried inside restricted ambients by allowed ambients, whence the "rogue" ambient can use an *out*-capability and thus end up immediately inside a restricted ambient. Another possibility is for the allowed ambient that carried the "rogue" to be opened and thereby setting the "rogue" free inside a restricted ambient. While 'respect' in general is not concerned with a "rogue" ambient being carried inside a restricted ambient by an allowed ambient, neither of the above mentioned scenarios are possible in a system that has the 'respect' property. In effect 'respect' states which ambients are allowed as immediate constituents of which other ambients.

If a stricter version of 'allowed interaction' is desired, disallowing "rogue" ambients to ever enter a restricted ambients no matter how, the notion of 'respect' is easily adapted to reflect this: use I^* instead of I in the second condition of the definition. In Section 5.3.4 we elaborate on these points, where other variations of 'allowed interaction' and 'respect' are also discussed.

In the following section the semantic soundness of Definition 5.13 is proved and a simple algorithm to check for 'respect' is briefly discussed.

Example 5.14 (Example 3.2 revisited) In this example we show how respect and allowed interaction can be used to guarantee that a given attacker can not enter a given firewall. We first define the firewall process and for this purpose we shall use the *Firewall* process from Example 3.2 annotated with markers and slightly rewritten to avoid notational clutter:

$$\mathit{Firewall} \;\triangleq\; (\nu_\mu w) w_\mu^{\ell^a_w}[k_{\mu_1}^{\ell^a_k}[\mathit{out}^{\ell^t_1} w_\mu.\mathit{in}^{\ell^t_2} m_{\mu_2}.\mathit{in}^{\ell^t_3} w_\mu] \mid \mathit{open}^{\ell^t_4} m_{\mu_2}.\mathit{open}^{\ell^t_5} n_{\mu_3}]$$

Furthermore, the processes P and Q of Example 3.2 are assumed to be the inactive process, $\mathbf{0}$, for clarity. We next define the attacker; for the purpose of this example a very primitive attacker is chosen:

$$\mathit{Atk} \;\triangleq\; a_{\mu_4}^{\ell^a_a}[\mathit{open}^{\ell^t_6} k_{\mu_1}.n_{\mu_3}^{\ell^a_n}[\mathbf{0}]]$$

This attacker resembles the (legitimate) *Agent* process of Example 3.2 but it knows only two of the passwords, namely k_{μ_1} and n_{μ_3} and not the third: m_{μ_2}.

For convenience we define:

$$\mathsf{FW} \;\triangleq\; \mathit{Firewall} \mid \mathit{Atk}$$

We are interested in using respect and allowed interaction to guarantee that the attacker will never enter the firewall. For this we first need an analysis of FW: let $me = [k \mapsto \mu_1, m \mapsto \mu_2, n \mapsto \mu_3, a \mapsto \mu_4]$, and note that me is compatible with FW. Furthermore, let $\ell \in \mathbf{Lab}^a$ and define:

$$\begin{aligned} I = \{&(\ell,\ell^a_a),(\ell,\ell^a_w),(\ell,\ell^a_k),(\ell^a_a,\ell^a_n),(\ell^a_a,\ell^t_6),\\ &(\ell^a_w,\ell^a_k),(\ell^a_w,\ell^t_4),(\ell^a_w,\ell^t_5),(\ell^a_k,\ell^t_1),(\ell^a_k,\ell^t_2),(\ell^a_k,\ell^t_3)\}\end{aligned}$$

and

$$H = \{(\ell_w^a, \mu), (\ell_k^a, \mu_1), (\ell_m^a, \mu_2), (\ell_n^a, \mu_3)(\ell_a^a, \mu_4)\}$$

Now it is easily seen that the following holds

$$\beta_{me}^{\ell}(\mathsf{FW}) \sqsubseteq (I, H) \ \wedge \ (I, H) \models_{me} \mathsf{FW}$$

From this we can easily use the static notion of respect to check if the attacker can ever enter the firewall; choose $\mathcal{R}_{in} = \{\ell_w^a\}$ stating that the firewall ambient, w_μ, is restricted. In this simple example no agents are allowed to enter the firewall and therefore we should choose $\mathcal{A}_{in}$ to be the empty set; but because of the way the firewall works, with a pilot ambient, k_{μ_1}, that is supposed to carry agents (and potential attackers) inside the firewall for further "password testing" we must allow this pilot to re-enter, hence we set $\mathcal{A}_{in} = \{\ell_k^a\}$. With these definitions in place, it is trivial to verify that FW respects $(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ since

$$\{\ell'' \in \mathbf{Lab}^a \,|\, (\ell', \ell'') \in I \wedge \ell' \in \mathcal{R}_{in}\} = \{\ell_k^a\} \subseteq \mathcal{A}_{in}$$

An important observation to make is that the above guarantee only holds for that *particular* attacker and that *particular* firewall, thus it can not be seen as a validation of the firewalls safety properties in general; however, this is the subject of Section 5.5. $\diamond$

5.3.3 A sound algorithm to check for 'allowed interaction'

In the previous sections the dynamic notion of 'allowed interaction' and the static notion of 'respect' were defined. It was observed that in systems having the dynamic property, the safety of critical processes could be guaranteed, in that only a predetermined set of processes were allowed to interact with the critical processes. In order to statically determine if a process has this property, a corresponding static notion, 'respect', was identified. We now need to establish that systems having the 'respect' property also have the 'allowed interaction' property, hence the following Proposition:

Proposition 5.15 *Let $P \in \mathbf{Proc}$ and $\mathcal{R}_{in}, \mathcal{A}_{in} \subseteq \mathbf{Lab}^a$. Now, given $me \in \mathbf{MEnv}$ and $\ell \in \mathbf{Lab}^a$, if P* respects *$(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ then P engages only in* allowed interaction *with respect to $(\mathcal{A}_{in}, \mathcal{R}_{in})$.*

Proof. Let $P \in \mathbf{Proc}$, $me \in \mathbf{MEnv}$, $\ell \in \mathbf{Lab}^a$ and $\mathcal{R}_{in}, \mathcal{A}_{in} \subseteq \mathbf{Lab}^a$ such that P *respects* $(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ and

$$P \to^* Q \to R$$

and the base step of $Q \to R$ is on the form:

$$n_{\mu}^{\ell_n^a}[in^{\ell^t} w_{\mu'}.S \mid T] \mid w_{\mu'}^{\ell_w^a}[U] \to w_{\mu'}^{\ell_w^a}[n_{\mu}^{\ell_n^a}[S \mid T] \mid U]$$

Now assume that $\ell_w^a \in \mathcal{R}_{in}$. Since P respects $(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ we have that there exists I and H, such that

$$\beta_{me}^{\ell}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\{\ell' \in \mathsf{Lab}^a \mid (\ell, \ell') \in I \wedge \ell \in \mathcal{R}_{in}\} \subseteq \mathcal{A}_{in} \tag{5.4}$$

Now it follows from Theorem 4.17 (subject reduction) that

$$\beta_{me}^{\ell}(R) \sqsubseteq (I, H) \wedge (I, H) \models_{me} R$$

Hence from Fact 5.7 we have that for some $\ell' \in \mathsf{Lab}^a$ and $me' \in \mathsf{MEnv}$:

$$\beta_{me'}^{\ell'}(w_{\mu'}^{\ell_w^a}[n_{\mu}^{\ell_n^a}[S \mid T] \mid U]) \sqsubseteq (I, H)$$

This implies that $(\ell_w^a, \ell_n^a) \in I$ and thus, since $\ell_w^a \in \mathcal{R}_{in}$, from (5.4) that $\ell_n^a \in \mathcal{A}_{in}$. Thus P only engages in *allowed interaction* with respect to $(\mathcal{A}_{in}, \mathcal{R}_{in})$. ∎

A simple algorithm to check for 'allowed interaction' can easily be derived from Proposition 5.15. Using the methods described in Chapter 6 an analysis, in this case the least, is obtained, which by definition fulfils the first condition of Definition 5.13. This analysis is then used to check the second condition in Definition 5.13. It is evident that, since the analysis obtained is the least, if the second condition is not met it can not possibly be met by any other analysis. If on the other hand the condition is fulfilled the process under study has the property of 'respect' and hence by the above Proposition it also has the property of 'allowed interaction'. An alternative approach to automatically checking for 'respect' is described in Section 5.3.4.

Finally while the notions of 'allowed interaction' and 'respect' imposes limitations on which ambients may enter which other ambients, the notions are easily extended to cope with capabilities in addition to ambients.

5.3.4 Further notions of 'respect'

In this section we will describe and discuss a few alternatives to the notion of 'respect', cf. Definition 5.13. We do this partly to indicate the scope of possible security properties that can be expressed in this framework, and partly to emphasise the role of the control flow analysis in the framework. The discussion in this section will be mostly of an informal nature, and very few formal results will be stated. Moreover, since static notions in general can be used to guarantee a wide variety of corresponding dynamic notions, the focus in the following discussion is on the static notions. The dynamic counterparts will only be commented upon briefly.

Facilitating automatic verification

Rather than computing the least solution, as proposed in the previous section, it is possible to weaken the precision of notion of 'respect' in a way that makes it easier, and faster, to verify automatically. The imprecision is obtained by placing stricter demands on the acceptable analyses, in particular: rather than using inequality, $\subseteq$, equality is used. Hence we define the notion of *strict respect*:

Definition 5.16 (Strict Respect) *Let $P \in \textsf{Proc}$, $\ell \in \textsf{Lab}^a$ and $me \in \textsf{MEnv}$ and let $\mathcal{R}_{in}, \mathcal{A}_{in} \subseteq \textsf{Lab}^a$. The process P* strictly respects *$(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ iff me is compatible with P and there exists I and H such that*

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\{\ell' \in \textsf{Lab}^a \mid (\ell, \ell') \in I \wedge \ell \in \mathcal{R}_{in}\} = \mathcal{A}_{in}$$

and

$$\{\ell' \in \textsf{Lab}^a \mid (\ell, \ell') \in I \wedge \ell \notin \mathcal{R}_{in}\} = \textsf{Lab}^a$$

We have the following relationship between the above definition and Definition 5.13:

Fact 5.17 *Strict Respect implies Respect*

Proof. Trivial, since $\{\ell' \in \textsf{Lab}^a \mid (\ell, \ell') \in I \wedge \ell \in \mathcal{R}_{in}\} = \mathcal{A}_{in} \subseteq \mathcal{A}_{in}$ ∎

Although the final condition in Definition 5.16 may seem superfluous it does enable us to *construct* an analysis rather than solve for it: it should be obvious from the definition that for any analysis, (I, H), that fulfils the conditions, the I component must be on the form:

$$I = (\mathcal{R}_{in} \times \mathcal{A}_{in}) \cup ((\textsf{Lab}^a \setminus \mathcal{R}_{in}) \times \textsf{Lab}^a)$$

Thus, instead of solving for the I component it can be *constructed*. Likewise, the H component can easily be calculated independently, as shown in Chapter 6. All that remains is to check if the constructed (I, H) is an acceptable analysis. The above fact merely establishes that the defined notion is sufficient for 'allowed interaction'.

Strong Respect

As indicated by the comments following Definition 5.13, it may be a desirable to verify that no "rogue" ambients are ever carried inside a restricted ambient. The following definition ensures that only explicitly allowed ambients will ever end up inside a restricted ambient. This follows from the fact that where I is concerned only with immediate constituents, I^* can be seen as a transitively closed version of I.

Definition 5.18 (Strong Respect) *Let $P \in \mathsf{Proc}$, $\ell \in \mathsf{Lab}^a$ and $me \in \mathsf{MEnv}$ and let $\mathcal{R}_{in}, \mathcal{A}_{in} \subseteq \mathsf{Lab}^a$. The process P* strongly respects *$(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ iff me is compatible with P and there exists I and H such that*

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\{\ell' \in \mathsf{Lab}^a \,|\, (\ell, \ell') \in I^* \wedge \ell \in \mathcal{R}_{in}\} \subseteq \mathcal{A}_{in}$$

As suggested by the name, the following relationship holds:

Fact 5.19 *Strong Respect implies Respect*

Proof. Follows from the inclusions:

$$\{\ell' \in \mathsf{Lab}^a \,|\, (\ell, \ell') \in I \wedge \ell \in \mathcal{R}_{in}\} \subseteq \{\ell' \in \mathsf{Lab}^a \,|\, (\ell, \ell') \in I^* \wedge \ell \in \mathcal{R}_{in}\} \subseteq \mathcal{A}_{in}$$

The first inclusion follows trivially from the definition of I^*. ■

We shall not go into further detail here, but the general technique is used in Section 5.3.5 for adding security classifications to ambients.

Controlling *in*-capabilities

This final variation of 'respect' is intended as an indicator of how the notion of 'respect' could have been formulated in a manner analogous to that of Definition 5.6, i.e. in a way that seeks to control the use of *in*-capabilities. As mentioned previously, this is in contrast to the rather more direct approach of Definition 5.13:

Definition 5.20 (Indirect Respect) *Let $P \in \mathsf{Proc}$, $\ell \in \mathsf{Lab}^a$ and $me \in \mathsf{MEnv}$ and let $\mathcal{R}_{in} \subseteq \mathsf{Lab}^a \times \mathsf{Mar}$, $\mathcal{A}_{in} \subseteq \mathsf{CAPA}$. The process P* indirectly respects *$(\mathcal{A}_{in}, \mathcal{R}_{in}); \ell, me$ iff me is compatible with P and there exists I and H such that*

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\forall (\ell^a, \mu) \in \mathcal{R}_{in} : \left\{ \mathsf{in}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \,\middle|\, \exists \ell^{a'} : (\ell^a, \ell^{a'}) \in II^{-1} \wedge (\ell^{a'}, \ell^t) \in I \right\} \subseteq \mathcal{A}_{in}$$

This alternative static notion is more precise in the case where two ambients (in different scopes) have the same label, although this comes at the cost of using markers and labels instead of merely labels.

For completeness sake we state the semantic soundness of the definition:

Proposition 5.21 *Indirect Respect implies Allowed Interaction*

Proof. Analogous to the proof of Proposition 5.8. ■

5.3.5 Adding Security Levels

In this section we show how to assign security clearances, or security levels, to ambients and how to use these to restrict access to ambients. The intuition being that an ambient with a given security clearance cannot enter an ambient with a higher security clearance. This is somewhat reminiscent of the lattice based security models, cf. [Lan81, Den76], where information is allowed only to flow from "low security" to "high security"; this is at the base of [DD77] and [BDNN99]: both describe how to statically guarantee that execution of a program does not violate such a security model.

Assume that we have a set of *security levels*:

$$\mathcal{L} = \{L_0, \ldots, L_n\}$$

In what follows the situation is sufficiently simple to not require that the security levels form a lattice, whence we assume that the security levels are ordered in the following way

$$L_0 \sqsubseteq \cdots \sqsubseteq L_n$$

In this presentation only ambients have security clearances and rather than introducing new syntactic constructs, we introduce the LEVEL function that assigns to each ambient label a security level.

$$\mathrm{LEVEL} : \textsf{\textbf{Lab}}^a \rightarrow \mathcal{L}$$

Note that this simple assignment is not able to distinguish between different ambients with the same label, thus ambients with the same label are assigned the same security clearance. This could be improved upon by assigning security clearances to pairs of ambient labels and markers, i.e. assigning security levels to ambient labels for ambients in a particular scope. The simple version will suffice for our purposes.

In order to use the previously defined notions of 'allowed interaction' and 'respect' we define for each security level, L, a set of restricted ambients, $\mathcal{R}^L_{in}$, and a corresponding set of allowed ambients, $\mathcal{A}^L_{in}$:

$$\mathcal{R}^L_{in} = \{\ell^a \in \textsf{\textbf{Lab}}^a \mid L = \mathrm{LEVEL}(\ell^a)\}$$

and

$$\mathcal{A}^L_{in} = \{\ell^a \in \textsf{\textbf{Lab}}^a \mid L \sqsubseteq \mathrm{LEVEL}(\ell^a)\}$$

Intuitively $\mathcal{R}^L_{in}$ represents the fact that the ambients with security clearance L are restricted at that level, and only ambients with a security clearance equal to or greater than L are allowed to enter ambients of level L, hence $\mathcal{A}^L_{in}$.

Now if for every $L \in \mathcal{L}$ we have that the 'allowed interaction' property holds, with respect to $(\mathcal{A}^L_{in}, \mathcal{R}^L_{in})$, no ambient will ever enter an ambient with a higher security clearance. By Proposition 5.15 it can statically be checked if a given system respects the assigned security levels. Note that while the dynamic property of 'allowed interaction' does not capture the situation where an ambient with a given security clearance is inside an ambient with a higher clearance initially, the static notion of 'respect' will not accept processes where this is the case. If further restrictions are needed the notions defined in Section 5.3.4 can be used; of particular relevance is 'strong respect' that would enable one to verify that no ambient of a lower clearance could ever end up inside an ambient of a higher clearance, no matter how deeply nested. Also note that in certain situations it might be desirable to allow ambients of higher clearance to carry ambients with lower clearances, thus in effect (temporarily) vouching for the integrity of the ambient with lower clearance. In such situations it would be advantageous to also check/control the use of both *open*-capabilities and *out*-capabilities, in a manner that also respects the assigned security clearances. The notions defined in Section 5.2 and Section 5.4 can be extended/applied in an obvious manner, similar to the above, to take security clearances into account.

5.4 No leaks and confinement

With the notions of 'allowed interaction' and 'respect' in place to ensure that only a predetermined set of distinguished ambients can enter certain, likewise distinguished, ambients, a natural question arises: once a distinguished ambient has entered another (distinguished) ambient, does it ever leave again? The question is equally valid in a setting where no ambients are special or distinguished, at least in the sense of Section 5.3, and it is this question that we will examine in this section. Apart from being a natural follow-up question, it has interesting and important applications, e.g. guaranteeing that a process does not "leak" information.

Before we proceed, consider the following expression:

$$n_{\mu}^{\ell_n^a}[in^{\ell_1^t} s_{\mu'}.out^{\ell_2^t} s_{\mu'}] \mid m_{\mu''}^{\ell_m^a}[in^{\ell_3^t} s_{\mu'}] \mid s_{\mu'}^{\ell_s^a}[\mathbf{0}]$$

We would like to be able to guarantee, that only the "good" ambient, $n_{\mu}^{\ell_n^a}$, can leave the server process, $s_{\mu'}^{\ell_s^a}$, and thus that the "malicious" ambient, $m_{\mu''}^{\ell_m^a}$ cannot leave the server once it has entered it.

Since an ambient can only enter a sibling ambient, and since the analysis does not record the (temporal) order in which capabilities are used, the analysis cannot be used to give the above guarantee in the same manner as in Section 5.3. Instead we proceed as in Section 5.2. As the developments in this section are very similar to those of Section 5.2, we shall only briefly discuss the notions defined here.

First we need a dynamic notion that captures the event of one ambient leaving another. This is done in a straightforward manner by controlling the *out*-capabilities:

Definition 5.22 (No Leaks) *Let* $P \in \mathsf{Proc}$*,* $\mathcal{A}_{out} \subseteq \mathsf{CAPA}$ *and let* $\mathcal{R}_{out} \subseteq \mathsf{Lab}^a \times \mathsf{Mar}$*. Process* P *has* no leaks wrt. $(\mathcal{A}_{out}, \mathcal{R}_{out})$ *iff* $\forall Q, R \in \mathsf{Proc}$ *such that*

$$P \to^* Q \to R$$

and the base step of $Q \to R$ *is on the form:*

$$m_{\mu}^{\ell_m^a}[n_{\mu'}^{\ell_n^a}[out^{\ell^t} m_\mu.S \mid T] \mid U] \to m_{\mu}^{\ell_m^a}[U] \mid n_{\mu'}^{\ell_n^a}[S \mid T]$$

then $(\ell_m^a, \mu) \in \mathcal{R}_{out} \Rightarrow \mathsf{out}^{\ell^t}\mu \in \mathcal{A}_{out}$.

The defined notion has been given the name 'no leaks' because it is an attempt at making sure that certain ambients, those that do not have an *out*-capability represented in $\mathcal{A}_{out}$, will never leave any ambient represented in $\mathcal{R}_{out}$. There is one caveat though, 'no leaks' will only ensure that an ambient does not leave by its own accord, i.e. by using an *out*-capability. It can still be carried out by an ambient that is allowed to leave. This is entirely parallel to the problems commented upon following Definition 5.12 and Definition 5.13. Before we discuss possible solutions, we need to define the static counterpart to 'no leaks':

Definition 5.23 (Confinement) *Let* $P \in \mathsf{Proc}$*,* $\mathcal{A}_{out} \subseteq \mathsf{CAPA}$*,* $\mathcal{R}_{out} \subseteq \mathsf{Lab}^a \times \mathsf{Mar}$ *and let* $\ell \in \mathsf{Lab}^a$ *and* $me \in \mathsf{MEnv}$*. Process* P *is said to be* confined wrt. $(\mathcal{A}_{out}, \mathcal{R}_{out}); \ell, me$ *iff* me *is compatible with* P *and there exists* I *and* H *such that*

$$\beta_{me}^{\ell}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

and

$$\forall (\ell^a, \mu) \in \mathcal{R}_{out} : \left\{ \mathsf{out}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P) \,\middle|\, (\ell^a, \ell^t) \in II \right\} \subseteq \mathcal{A}_{out}$$

Since the above definition is straightforward and should present no surprises, we immediately turn towards establishing the semantic soundness of the defined notions.

Proposition 5.24 *Let* $P \in \mathsf{Proc}$*,* $\ell \in \mathsf{Lab}^a$*,* $me \in \mathsf{MEnv}$*,* $\mathcal{A}_{out} \subseteq \mathsf{CAPA}$ *and let* $\mathcal{R}_{out} \subseteq \mathsf{Lab}^a \times \mathsf{Mar}$*. If* P *is confined wrt.* $(\mathcal{A}_{out}, \mathcal{R}_{out}); \ell, me$ *then* P *has no leaks wrt.* $(\mathcal{A}_{out}, \mathcal{R}_{out})$

Proof. Analogous to the proof of Proposition 5.8. We have the following situation:

$$P \to^* Q \to R$$

With the basestep of $Q \to R$ on the form below

$$m_{\mu}^{\ell_m^a}[n_{\mu'}^{\ell_n^a}[out^{\ell^t} m_\mu.S \mid T] \mid U] \to m_{\mu}^{\ell_m^a}[U] \mid n_{\mu'}^{\ell_n^a}[S \mid T] \tag{5.5}$$

Since P is confined wrt. $(\mathcal{A}_{out}, \mathcal{R}_{out}); \ell, me$ we have that there exists I and H such that

$$\beta^{\ell}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$$

And from Theorem 4.17 (subject reduction) it follows that:

$$\beta^{\ell}_{me}(Q) \sqsubseteq (I, H) \wedge (I, H) \models_{me} Q$$

Now using Fact 5.7 with (5.5) we have that, for some $me' \in \mathbf{MEnv}$ and some $\ell' \in \mathsf{Lab}^a$:

$$\beta^{\ell'}_{me'}(m^{\ell^a_m}_{\mu}[n^{\ell^a_n}_{\mu'}[out^{\ell^t} m_{\mu}.S \mid T] \mid U]) \sqsubseteq (I, H)$$

In particular: $(\ell^a_m, \ell^t) \in II$. Moreover, since $\mathsf{out}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P)$ it follows from Definition 5.6 that $\mathsf{out}^{\ell^t}\mu \in \mathcal{A}_{out}$. This ends the proof. ∎

As was the case for 'respect' the static notion of 'confinement' does not capture the situation of an ambient being carried inside another ambient. The main reason for this is that 'no leaks' is mainly concerned with controlling the *out*-capabilities, rather than which ambients that uses them. One solution to this would be to combine 'no leaks' with 'respect' (or one of its variants). The former to control which *out*-capabilities are used and the latter to verify that only the "right" ambients can be inside ambients that leave. We shall not go into further detail with this approach.

Example 5.25 Returning to the expression at the beginning of this section, we define for convenience:

$$\mathsf{S} = n^{\ell^a_n}_{\mu}[in^{\ell^t_1} s_{\mu'}.out^{\ell^t_2} s_{\mu'}] \mid m^{\ell^a_m}_{\mu''}[in^{\ell^t_3} s_{\mu'}] \mid s^{\ell^a_s}_{\mu'}[\mathbf{0}]$$

We first need to establish an analysis for S; we define a marker environment, $me = [n \mapsto \mu, m \mapsto \mu'', s \mapsto \mu']$, easily seen to be compatible with S. Next let $\ell \in \mathsf{Lab}^a$ and define I and H:

$$\begin{aligned} I = \{&(\ell, \ell^a_n), (\ell, \ell^a_m), (\ell, \ell^a_s), (\ell^a_n, \ell^t_1), (\ell^a_n, \ell^t_2), \\ &(\ell^a_m, \ell^t_3), (\ell^a_s, \ell^a_n), (\ell^a_s, \ell^a_m)\} \end{aligned}$$

and

$$H = \{(\ell^a_n, \mu), (\ell^a_m, \mu''), (\ell^a_s, \mu')\}$$

The following is easily established:

$$\beta^{\ell}_{me}(\mathsf{S}) \sqsubseteq (I, H) \;\wedge\; (I, H) \models_{me} \mathsf{S}$$

Next we choose $\mathcal{R}_{out} = \{(\ell^a_s, \mu')\}$ and $\mathcal{A}_{out} = \left\{\mathsf{out}^{\ell^t_2}\mu'\right\}$; and thus we can establish that S is confined with respect to $(\mathcal{A}_{out}, \mathcal{R}_{out}); \ell, me$ since

$$\forall (\ell^a, \mu) \in \mathcal{R}_{out} : \left\{\mathsf{out}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P) \,\middle|\, (\ell^a, \ell^t) \in II\right\} = \left\{\mathsf{out}^{\ell^t_2}\mu'\right\} \subseteq \mathcal{A}_{out}$$

From the I component of the analysis above we can furthermore see that only the ambient n_{μ} can possibly contain the capability labelled ℓ^t_2 (since (ℓ^a_n, ℓ^t_2) is the only pair containing ℓ^t_2 as a second component); hence only n_{μ} can move out from the server, $s_{\mu'}$. ◇

5.5 Validating Firewalls

In this section we sketch how the control flow analysis of Chapter 4 can be used to guarantee the protectiveness of a proposed firewall (a firewall in the sense of [CG97a, CG98]). The technical details can be found in [NNHJ99].

In [CG98] a firewall, where only agents knowing the correct passwords should be able to enter, is modelled in the ambient calculus; furthermore it is shown that any agent on a special form will indeed enter the firewall. Here, and in [NNHJ99], we are interested in examining if the converse holds for a proposed firewall, namely that agents that do not know the correct passwords, called *attackers*, are *not* allowed to enter.

The main idea is to construct an attacker, called the *hardest attacker*, that is as hard to protect against as any other attacker. Then, using the analysis, checking if the hardest attacker can enter the firewall; this is done by analysing the firewall and the hardest attacker in parallel. Using the analysis, it is determined if the hardest attacker can not enter the firewall; if this is the case then, by definition of the hardest attacker, no other attacker can enter and the firewall is said to be *protective* (with respect to the correct passwords). Again we refer to [NNHJ99] for the details.

A point worth noting is that the hardest attacker is constructed by a careful exploitation of the detailed operation of the control flow analysis. Another point worth noting is that such a check as described above only guarantees that processes that do not know the correct passwords can not enter. If an attacker has (partial) knowledge of one or more passwords no guarantee can be given, although there is ongoing work to lift this restriction. Finally, note that the notion of firewall studied here and in [CG98, NNHJ99] is a somewhat idealised firewall (see [Com95] for details of "real" firewalls).

Note that in the following examples we use the syntax of [NNHJ99] rather than the one used in the previous sections in this chapter.

Example 5.26 In [NNHJ99] the following process in the ambient calculus is validated as being a *protective* firewall with respect to the passwords: k_0, k_1 and k_2:

$$(\nu_\mu w)w^{\ell^a_w}[(k_0^{\ell^a_{k_0}}[out^{\ell^t_1} w.in^{\ell^t_2} k_1.in^{\ell^t_3} w] \mid open^{\ell^t_4} k_1.open^{\ell^t_5} k_2.P)]$$

Note that the process has rewritten to fit the syntax defined in Section 3.2. Note that the above process is very similar to the process *Firewall* of Example 3.2. The process below is a typical agent that knows the correct passwords:

$$k_1^{\ell^a_{k_1}}[open^{\ell^t_6} k_0.k_2^{\ell^a_{k_2}}[Q]]$$

It is easily shown that this agent can enter the firewall. $\diamond$

Example 5.27 The next proposed firewall is also from [NNHJ99], but this firewall contains a *trapdoor*:

$$(\nu_\mu) w^{\ell_w^a}[k_0^{\ell_{k_0}^a}[out^{\ell_1^t} w.in^{\ell_2^t} k_1.in^{\ell_3^t} w] \mid open^{\ell_4^t} k_1.open^{\ell_5^t} k_2.P \\ \mid t^{\ell_t^a}[out^{\ell_6^t} w.in^{\ell_7^t} w.open^{\ell_8^t} q] \mid open^{\ell_9^t} t]$$

The trapdoor can be used by an attacker that does *not* know the correct passwords, such as the following:

$$q^{\ell_q^a}[in^{\ell_{10}^t} t]$$

It is easily shown that q can enter the above firewall, even though q does not know the passwords, hence the firewall should *not* be validated as protective and this is indeed the result of the validation procedure described above, cf. [NNHJ99]. $\diamond$

Chapter 6

Finding Least Solutions

In Chapter 4 it was shown how to check whether a proposed analysis was acceptable with respect to a given process. Subsequently it was shown that all processes admits a least analysis by the so-called Moore-family result. In this chapter a technique to finding these least analyses will be presented. The overall idea is to produce *set constraints* that can be solved using an independent *constraint solver*. This separation between specification (producing set constraints) and implementation (solving the constraints) is one of the main advantages of using set constraints. In the literature [Aik94, HJ94, AKVW94], systems of set constraints with varying expressive power have been presented. The idea is that you choose the system of constraint primitives you need and a system implementing it. Several such systems exists (e.g. BANE [AFFS98]) which incorporates several different constraint systems with different trade-offs between expressive power and complexity. The constraints generated from the analysis specification of Chapter 4 are so simple that an algorithm to solve them is fairly small, so we have chosen to present it here rather than relying on an existing system. Set constraints have been used for a variety of problems, for instance typing in OO-languages [PS94] and analysis of ML [Hei94].

In this chapter we will first produce constraints that relate very closely to the analysis of Table 4.1 and 4.2. These constraints are syntactic entities and a suitable semantic interpretation of them is presented. Following this we prove that the acceptable analyses to a given process are solutions to the constraints generated from the process and vice versa. An algorithm that solves the constraints is then presented and proved to produce the least solution and it is shown that the least analysis to a process can be found in time $O(s^5)$ assuming that the size of the process is s. Following this comes a presentation of an alternative scheme for producing constraints which leads to an improvement on the bound, showing that the least analysis can be found in time $O(s^4)$.

6.1 Problem setting

The aim of this chapter is to find the least analysis to a given process $P_\star$ with an enclosing ambient $\ell^a_\star$ and an initial marker environment $me_\star$ where the free names of $P_\star$ are bound, i.e. $fn(P_\star) \subseteq \mathrm{dom}(me_\star)$. Reformulated, the aim is to find the least analysis (I, H) to

$$\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I, H) \;\wedge\; (I, H) \models_{me_\star} P_\star$$

with respect to the ordering introduced in Chapter 4:

$$(I_1, H_1) \sqsubseteq (I_2, H_2) \text{ iff } I_1 \subseteq I_2 \;\wedge\; H_1 \subseteq H_2$$

The existence of such a least analysis relies on Theorem 4.23 (Moore-family result). We divide the task of obtaining the least analysis in two stages. Section 6.2 shows that it is straightforward to compute the least H-component. Based on this information we generate constraints for the I-component and by solving these we obtain the least solution.

6.2 Computing the least H-component

Recall that the H-component of an analysis is responsible for recording the association between restrictions (represented by markers) and ambients (represented by ambient labels), i.e. if an ambient labelled ℓ^a is bound at a restriction μ then $(\ell^a, \mu) \in H$ should hold for an analysis to be acceptable. New associations between markers and ambient labels do not arise under reduction (note that $!P$ does not change this fact since $!P$ represents any number of *exact* copies of P, i.e. the markers and labels are the same in the different copies), thus making it quite simple to obtain the least H-component by recording the association between the markers and the ambient labels in $P_\star$ (using $me_\star$ for the free names of $P_\star$); this is exactly what the representation function β does and the following shows that the least H-component is equal to the second component of $\beta^{\ell^a_\star}_{me_\star}(P_\star)$: from $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I, H)$ it follows that the least H-component is at least as great as the second component of $\beta^{\ell^a_\star}_{me_\star}(P_\star)$; equality follows from the following proposition.

Proposition 6.1 *For all I, H, me and P we have*

$$\textit{If } (I, H) \models_{me} P \textit{ then } \forall H' \subseteq H : (I, H') \models_{me} P$$

Proof. By structural induction on P, with the following induction hypothesis:

$$\begin{array}{l} \forall I, H, me : \\ \quad \textit{If } (I, H) \models_{me} P \textit{ then } \forall H' \subseteq H : (I, H') \models_{me} P \end{array}$$

Case (in ($P = in^{\ell^t} n.Q$)): Using the definition of $\models$ we have the following:

$$\begin{array}{ll}(1) & (I,H) \models_{me} Q \\ (2) & \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : (((\ell^a, \ell^t) \in I \ \wedge \ (\ell^{a''}, \ell^a) \in I \\ & \quad \wedge \ (\ell^{a''}, \ell^{a'}) \in I \ \wedge \ (\ell^{a'}, me(n)) \in H \Rightarrow (\ell^{a'}, \ell^a) \in I)))\end{array}$$

Applying the induction hypothesis to (1) we have that

$$\forall H' \subseteq H : (I, H') \models_{me} Q$$

and from (2) we have that

$$\begin{array}{l}\forall H' \subseteq H : \forall \ell^a, \ell^{a'}, \ell^{a''} : (((\ell^a, \ell^t) \in I \ \wedge \ (\ell^{a''}, \ell^a) \in I \\ \quad \wedge \ (\ell^{a''}, \ell^{a'}) \in I \ \wedge \ (\ell^{a'}, me(n)) \in H') \Rightarrow (\ell^{a'}, \ell^a) \in I))\end{array}$$

Combining this we obtain

$$\forall H' \subseteq H : (I, H') \models_{me} in^{\ell^t} n.Q$$

The remaining cases are similar, or trivial applications of the induction hypothesis. ■

We define $\mathcal{H}(me)[\![P]\!]$ to be the second component of $\beta_{me}^{\ell^a}(P)$ which does not depend on ℓ^a as stated by the following fact.

Fact 6.2 *Let $P \in$ **Proc** and $me \in$ **MEnv** be given. For two ambient labels $\ell^a, \ell^{a'} \in$ **Lab**^{a} we have that if $(I_{\ell^a}, H_{\ell^a}) = \beta_{me}^{\ell^a}(P)$ and $(I_{\ell^{a'}}, H_{\ell^{a'}}) = \beta_{me}^{\ell^{a'}}(P)$ for some $I_{\ell^a}, I_{\ell^{a'}}, H_{\ell^a}$ and $H_{\ell^{a'}}$ then $H_{\ell^a} = H_{\ell^{a'}}$*

Proof. By structural induction on P. ■

A simple calculation suffices to show that $\mathcal{H}$: **MEnv** $\to$ **Proc** $\to$ **HNam** can be computed as in Table 6.1.

Example 6.3 Consider once again the term from Example 3.1:

$$(\nu_\mu m)(n^{\ell^a_n}[in^{\ell^t_1} m] \mid m^{\ell^a_m}[open^{\ell^t_2} n])$$

Choosing $P_\star$ to be our example term and letting $me_\star = [n \mapsto \mu_n]$, we have:

$$\mathcal{H}(me_\star)[\![P_\star]\!] = \{(\ell^a_m, \mu), (\ell^a_n, \mu_n)\}$$

which is the H-component for the least analysis. ◇

$$
\begin{array}{lcl}
\multicolumn{3}{c}{\mathcal{H} : \textsf{MEnv} \to \textsf{Proc} \to \textsf{HNam}} \\
\mathcal{H}(me)[\![(\nu_\mu n)P]\!] & = & \mathcal{H}(me[n \mapsto \mu])[\![P]\!] \\
\mathcal{H}(me)[\![\mathbf{0}]\!] & = & \emptyset \\
\mathcal{H}(me)[\![P \mid Q]\!] & = & \mathcal{H}(me)[\![P]\!] \cup \mathcal{H}(me)[\![P]\!] \\
\mathcal{H}(me)[\![!P]\!] & = & \mathcal{H}(me)[\![P]\!] \\
\mathcal{H}(me)[\![n^{\ell^a}[P]]\!] & = & \mathcal{H}(me)[\![P]\!] \cup (\ell^a, me(n)) \\
\mathcal{H}(me)[\![in^{\ell^t} n.P]\!] & = & \mathcal{H}(me)[\![P]\!] \\
\mathcal{H}(me)[\![out^{\ell^t} n.P]\!] & = & \mathcal{H}(me)[\![P]\!] \\
\mathcal{H}(me)[\![open^{\ell^t} n.P]\!] & = & \mathcal{H}(me)[\![P]\!]
\end{array}
$$

Table 6.1: The function $\mathcal{H}$

6.3 Generating constraints

Now that we have shown how to compute the least H-component, it remains to find the least I-component. In this section we show that if a "sufficiently small" I is a solution to constraints generated from $P_\star$, $\ell^a_\star$, $me_\star$ and H (where $H \supseteq \mathcal{H}(me_\star)[\![P_\star]\!]$), then (I, H) is an acceptable analysis and vice versa. From this we conclude that we can find the least I-component by finding the least solution to the constraints generated from $P_\star$, $\ell^a_\star$, $me_\star$ and $\mathcal{H}(me_\star)[\![P_\star]\!]$.

In order to generate a finite number of constraints from $P_\star$, $\ell^a_\star$, $me_\star$ and H, we need only to consider the finite set of labels that occur in the process $P_\star$ of interest together with the enclosing ambient label, $\ell^a_\star$. Let $\textsf{Lab}^a_\star$ be the finite set of ambient labels occurring in the process $P_\star$ together with $\ell^a_\star$, likewise let $\textsf{Lab}^t_\star$ be the set of thread labels occurring in $P_\star$ and let $\textsf{Lab}_\star = \textsf{Lab}^a_\star \cup \textsf{Lab}^t_\star$ be the set of all labels occurring in $P_\star$ together with $\ell^a_\star$. Define $I^\top_\star$ and $H^\top_\star$ by

$$I^\top_\star = \left\{(\ell^a, \ell^{at}) \,\middle|\, \ell^a \in \textsf{Lab}^a_\star, \ell^{at} \in \textsf{Lab}_\star\right\}$$

$$H^\top_\star = \{(\ell^a, \mu) \mid \ell^a \in \textsf{Lab}^a_\star, \mu \in \textsf{Mar}\}$$

Fact 6.4 and Proposition 6.5 shows that $(I^\top_\star, H^\top_\star)$ is an acceptable analysis.

Fact 6.4 *We have that $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I^\top_\star, H^\top_\star)$.*

Proof. By structural induction on $P_\star$. ∎

Proposition 6.5 *For all me with $\mathrm{dom}(me) \subseteq fn(P_\star)$ we have $(I^\top_\star, H^\top_\star) \models_{me} P_\star$*

Proof. We show that $(I^\top_\star, H^\top_\star) \models_{me} P$ holds for all subprocesses P of $P_\star$ by structural induction on P using the induction hypothesis:

$$\forall me : (I^\top_\star, H^\top_\star) \models_{me} P$$

$$
\begin{array}{ll}
(res) & \mathcal{C}_\beta(\ell^a)[\![(\nu_\mu n)P]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \\[1em]
(zero) & \mathcal{C}_\beta(\ell^a)[\![\mathbf{0}]\!] = \emptyset \\[1em]
(par) & \mathcal{C}_\beta(\ell^a)[\![P \mid Q]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \mathcal{C}_\beta(\ell^a)[\![Q]\!] \\[1em]
(repl) & \mathcal{C}_\beta(\ell^a)[\![!P]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \\[1em]
(amb) & \mathcal{C}_\beta(\ell^a)[\![n^{\ell^a_n}[P]]\!] = \mathcal{C}_\beta(\ell^{a'})[\![P]\!] \cup \{\{l^a_n\} \subseteq \mathtt{I}(l^a)\} \\[1em]
(in) & \mathcal{C}_\beta(\ell^a)[\![in^{\ell^t} n.P]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \{\{l^t\} \subseteq \mathtt{I}(l^a)\} \\[1em]
(out) & \mathcal{C}_\beta(\ell^a)[\![out^{\ell^t} n.P]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \{\{l^t\} \subseteq \mathtt{I}(l^a)\} \\[1em]
(open) & \mathcal{C}_\beta(\ell^a)[\![open^{\ell^t} n.P]\!] = \mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \{\{l^t\} \subseteq \mathtt{I}(l^a)\}
\end{array}
$$

Table 6.2: Constraint Based Control Flow Analysis (Representation function)

Case (in ($P = in^{\ell^t} n.Q$)): By the induction hypothesis we have $(I_\star^\top, H_\star^\top) \models_{me} Q$. Now let $\ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a$ be given. Assume that

$$(\ell^a, \ell^t) \in I_\star^\top \;\wedge\; (\ell^{a''}, \ell^a) \in I_\star^\top \;\wedge\; (\ell^{a''}, \ell^{a'}) \in I_\star^\top \;\wedge\; (\ell^{a'}, me(n)) \in H_\star^\top$$

By the definition of $I_\star^\top$ and $H_\star^\top$ this means that $\ell^a, \ell^{a'} \in \mathbf{Lab}_\star^a$ and so $(\ell^{a'}, \ell^a) \in I_\star^\top$.

The remaining cases are similar, or trivial applications of the induction hypothesis. ■

Recalling Theorem 4.23 (Moore-family result), two consequences of this are:

- Each process $P_\star$ has an analysis (I, H) where $I \subseteq I_\star^\top$
- Each process $P_\star$ has a least analysis (I, H) where $I \subseteq I_\star^\top$

We will henceforth focus on I-components that only contain labels from $\mathbf{Lab}_\star$, i.e. $I \subseteq I_\star^\top$.

The two functions $\mathcal{C}_\beta$ and $\mathcal{C}_\models$ of Table 6.2 and Table 6.3 respectively generate the constraints to be solved and will be explained below. The functions are generating sets of constraints that correspond to the logical formulae that should hold in order for a proposed analysis to be acceptable.

Intuitively, the constraints generated by $\mathcal{C}_\beta(\ell_\star^a)[\![P_\star]\!]$ correspond to $\beta^{\ell_\star^a}_{me_\star}(P_\star) \sqsubseteq (I, H)$. As an example take $P_\star = n^{\ell^a_n}[Q]$:

$$\beta^{\ell_\star^a}_{me_\star}(n^{\ell^a_n}[Q]) \sqsubseteq (I, H) \iff (\ell_\star^a, \ell_n^a) \in I \wedge (\ell_n^a, me_\star(n)) \in H \wedge \beta^{\ell_n^a}_{me_\star}(Q) \sqsubseteq (I, H)$$

$$
\begin{array}{ll}
(res) & \mathcal{C}_{\models}(me,H)[\![(\nu_\mu n)P]\!] = \mathcal{C}_{\models}(me[n \mapsto \mu],H)[\![P]\!] \\[1em]
(zero) & \mathcal{C}_{\models}(me,H)[\![\mathbf{0}]\!] = \emptyset \\[1em]
(par) & \mathcal{C}_{\models}(me,H)[\![P \mid Q]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \cup \mathcal{C}_{\models}(me,H)[\![Q]\!] \\[1em]
(repl) & \mathcal{C}_{\models}(me,H)[\![!P]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \\[1em]
(amb) & \mathcal{C}_{\models}(me,H)[\![n^{\ell^a}[P]]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \\[1em]
(in) & \mathcal{C}_{\models}(me,H)[\![in^{\ell^t}n.P]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \cup \\
& \quad \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'}) \;\Big|\; \\
& \qquad \ell^a \in \mathbf{Lab}^a_\star, (\ell^{a'}, me(n)) \in H, \ell^{a''} \in \mathbf{Lab}^a_\star \Big\} \\[1em]
(out) & \mathcal{C}_{\models}(me,H)[\![out^{\ell^t}n.P]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \cup \\
& \quad \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \;\Big|\; \\
& \qquad \ell^a \in \mathbf{Lab}^a_\star, (\ell^{a'}, me(n)) \in H, \ell^{a''} \in \mathbf{Lab}^a_\star \Big\} \\[1em]
(open) & \mathcal{C}_{\models}(me,H)[\![open^{\ell^t}n.P]\!] = \mathcal{C}_{\models}(me,H)[\![P]\!] \cup \\
& \quad \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^{a'}) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^a) \;\Big|\; \\
& \qquad \ell^a \in \mathbf{Lab}^a_\star, (\ell^{a'}, me(n)) \in H, \ell^{at} \in \mathbf{Lab}_\star \Big\}
\end{array}
$$

Table 6.3: Constraint Based Control Flow Analysis (Closure property)

which should be compared to

$$\mathcal{C}_\beta(\ell^a_\star)[\![n^{\ell^a_n}[Q]]\!] = \{\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)\} \cup \mathcal{C}_\beta(\ell^a_n)[\![Q]\!]$$

First notice that no syntax representing the check $(\ell^a_n, me(n)) \in H$ is generated, as we are only generating constraints for I. The check $(\ell^a_\star, \ell^a_n) \in I$ is represented by the syntax, $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ whereas the "recursive" check, $\beta^{\ell^a_n}_{me_\star}(Q) \sqsubseteq (I, H)$, is represented by the constraints generated from $\mathcal{C}_\beta(\ell^a_n)[\![Q]\!]$.

The constraints generated by $\mathcal{C}_\models(me_\star, H)[\![P_\star]\!]$ correspond to $(I, H) \models_{me_\star} P_\star$. As an example take $P_\star = in^{\ell^t} n.Q$:

$$\begin{aligned}
&(I, H) \models_{me_\star} in^{\ell^t} n.Q \iff \\
&\quad (I, H) \models_{me} Q \ \wedge\ \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : (((\ell^a, \ell^t) \in I \ \wedge\ (\ell^{a''}, \ell^a) \in I\ \wedge \\
&\quad\quad (l^{a''}, l^{a'}) \in I \ \wedge\ (\ell^{a'}, me_\star(n)) \in H) \Rightarrow (\ell^{a'}, \ell^a) \in I)
\end{aligned}$$

which should be compared to

$$\begin{aligned}
&\mathcal{C}_\models(me_\star, H)[\![in^{\ell^t} n.Q]\!] = \mathcal{C}_\models(me_\star, H)[\![Q]\!] \cup \\
&\quad \Big\{\{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \,\Big| \\
&\quad\quad \ell^a \in \mathbf{Lab}^a_\star, (\ell^{a'}, me_\star(n)) \in H, \ell^{a''} \in \mathbf{Lab}^a_\star\Big\}
\end{aligned}$$

In this situation, the check $\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a : \ldots$ corresponds to a number of constraints, where the use of $\mathbf{Lab}^a_\star$ is justified by Fact 6.4 and Proposition 6.5. The "recursive" check, $(I, H) \models_{me_\star} Q$ corresponds to the constraints generated by $\mathcal{C}_\models(me_\star, H)[\![Q]\!]$. The use of `=>` in the constraints is justified by the fact that $((a_1 \ \wedge\ \cdots\ \wedge\ a_n) \Rightarrow b) \iff (a_1 \Rightarrow \cdots \Rightarrow a_n \Rightarrow b)$.

Remark 6.6 The check $(\ell^a, \ell^{at}) \in I$ is represented by $\{l^{at}\} \subseteq \mathtt{I}(\ell^a)$ where $\{l^{at}\}$ is a constant and $\mathtt{I}(l^a)$ is a variable to solve for. We could also have chosen to represent the check more directly, e.g. $(l^a, l^{at}) \in \mathtt{I}$, where (l^a, l^{at}) is a constant and $\mathtt{I}$ is the only variable to solve for, but this approach has the disadvantage that it does not work well with the worst case time complexity developed for the algorithm presented in Section 6.4. Remark 6.12 discusses how the worst case time complexity of solving the constraints depends on the choice of representation. ◇

Example 6.7 Let $P_\star$ be our example process from Example 6.3:

$$(\nu_\mu m)(n^{\ell^a_n}[in^{\ell^t_1} m] \mid m^{\ell^a_m}[open^{\ell^t_2} n])$$

Assuming $me_\star = [n \mapsto \mu_n]$ where μ and μ_n are distinct we obtain the following constraints from $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!]$:

$$\begin{gathered}
\{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \\
\{l^t_1\} \subseteq \mathtt{I}(l^a_n) \\
\{l^a_m\} \subseteq \mathtt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \mathtt{I}(l^a_m)
\end{gathered}$$

and the following constraints from $\mathcal{C}_\models(me_\star, \mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!]$:

$$\begin{array}{l}
\{l^t_1\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_1\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_\star) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_\star) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_n) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_n) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_n) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_\star\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^a_m\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \texttt{I}(l^a_m) \\
\{l^t_2\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^a_n\} \subseteq \texttt{I}(l^a_m) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \texttt{I}(l^a_m)
\end{array}$$

These constraints will be used in Example 6.11 to illustrate the workings of the constraint solving algorithm. ◇

The constraints generated by $\mathcal{C}_\beta$ and $\mathcal{C}_\models$ can be produced by the following grammar:

$$\begin{array}{rcl}
\mathbf{cnstr} & ::= & \mathbf{cst} \subseteq \mathbf{var} \mid \mathbf{cnstr} \texttt{ => } \mathbf{cnstr} \\
\mathbf{cst} & ::= & \{l^{at}\}, \ell^{at} \in \mathsf{Lab} \\
\mathbf{var} & ::= & \texttt{I}(l^a), \ell^a \in \mathsf{Lab}^a
\end{array}$$

In the following, syntax produced by this grammar will be given meaning. When parsing, `=>` is taken to be right-associative and for instance

$$\mathbf{cst}_1 \subseteq \mathbf{var}_1 \texttt{ => } \mathbf{cst}_2 \subseteq \mathbf{var}_2 \texttt{ => } \mathbf{cst}_3 \subseteq \mathbf{var}_3$$

is parsed as

$$\mathbf{cst}_1 \subseteq \mathbf{var}_1 \texttt{ => } (\mathbf{cst}_2 \subseteq \mathbf{var}_2 \texttt{ => } \mathbf{cst}_3 \subseteq \mathbf{var}_3)$$

It is important to stress that while $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge \ (I,H) \models_{me_\star} P_\star$ is a logical formula, $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_\models(me_\star, H)[\![P_\star]\!]$ is a set of syntactic entities. To give meaning to the syntax we translate the **cst** and **var** productions into a subset of **Lab**:

$$\begin{array}{rcl} I[\![\{l^{at}\}]\!] & = & \left\{\ell^{at}\right\} \\ I[\![\mathtt{I}(l^{a})]\!] & = & \left\{\ell^{at} \,\middle|\, (\ell^{a},\ell^{at}) \in I\right\} \end{array}$$

Next we define a satisfaction relation $\models_c$ on **cnstr**:

$$\begin{array}{rcl} I \models_c \mathbf{cst} \subseteq \mathbf{var} & \text{iff} & I[\![\mathbf{cst}]\!] \subseteq I[\![\mathbf{var}]\!] \\ I \models_c \mathbf{cnstr} \texttt{ => } \mathbf{cnstr} & \text{iff} & I \models_c \mathbf{cnstr} \Rightarrow I \models_c \mathbf{cnstr} \end{array}$$

We can lift the definition of $\models_c$ to a set C of constraints by

$$I \models_c C \quad \text{iff} \quad \forall cc \in C : I \models_c cc$$

In the following proofs we investigate the relationship between β and $\models$ on one side and $\mathcal{C}_\beta$ and $\mathcal{C}_\models$ on the other. These proofs use the following observation extensively: remember that `=>` is right-associative so

$$I \models_c \{l_1^{at}\} \subseteq \mathtt{I}(l_1^{a}) \texttt{ => } \cdots \texttt{ => } \{l_n^{at}\} \subseteq \mathtt{I}(l_n^{a}) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^{a})$$

if and only if

$$((\ell_1^{a},\ell_1^{at}) \in I \ \wedge\ \cdots\ \wedge\ (\ell_n^{a},\ell_n^{at}) \in I) \ \Rightarrow\ (\ell^{a},\ell^{at}) \in I$$

where $n \geq 1$.

The following proposition relates β and $\mathcal{C}_\beta$; the second conjunct on the right hand side is needed since $\mathcal{C}_\beta$ does not generate constraints for H.

Proposition 6.8 *For all I and H we have*

$$\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \iff (I \models_c \mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \ \wedge\ \mathcal{H}(me_\star)[\![P_\star]\!] \subseteq H)$$

Proof. We first prove the "$\Rightarrow$"-direction. By definition of $\mathcal{H}$ we have that if $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H)$ then $\mathcal{H}(me_\star)[\![P_\star]\!] \subseteq H$. The rest of the claim is proved by structural induction on $P_\star$, with the following induction hypothesis:

$$\begin{array}{l} \forall I, H, \ell^a, me : \\ \quad \beta^{\ell^a}_{me}(P) \sqsubseteq (I,H) \ \Rightarrow\ I \models_c \mathcal{C}_\beta(\ell^a)[\![P]\!] \end{array}$$

Case (amb ($P = n^{\ell^a_n}[Q]$)): By the definition of β it follows that $\beta^{\ell^a_n}_{me}(Q) \sqsubseteq (I,H)$ and that $(\ell^a,\ell^a_n) \in I$. Using the induction hypothesis and the definitions of $\mathcal{C}_\beta$ and $\models_c$ we have:

$$I \models_c \mathcal{C}_\beta(\ell^a_n)[\![Q]\!] \ \wedge\ I \models_c \{\{l^a_n\} \subseteq \mathtt{I}(l^a)\}$$

and we obtain $I \models_c \mathcal{C}_\beta(\ell^a)[\![n^{\ell^a_n}[Q]]\!]$.

The remaining cases are similar, or trivial applications of the induction hypothesis.

The "$\Leftarrow$"-direction is proved by structural induction on P, with the induction hypothesis:

$$\begin{array}{l}\forall I, H, \ell^a, me : \\ \quad I \models_c \mathcal{C}_\beta(\ell^a)[\![P]\!] \;\wedge\; \mathcal{H}(me)[\![P]\!] \subseteq H \;\Rightarrow\; \beta^{\ell^a}_{me}(P) \sqsubseteq (I, H)\end{array}$$

Case (amb ($P = n^{\ell^a_n}[Q]$)): We have

$$\begin{array}{ll}(1) & I \models_c \mathcal{C}_\beta(\ell^a)[\![n^{\ell^a_n}[Q]]\!] \\ (2) & \mathcal{H}(me)[\![n^{\ell^a_n}[Q]]\!] \subseteq H\end{array}$$

From (1) we get $I \models_c \left\{\{l^{a'}\} \subseteq \mathtt{I}(l^a)\right\}$ which by the definition of $\models_c$ gives us $(\ell^a, \ell^a_n) \in I$. From (2) we get $(\ell^a_n, me(n)) \in H$. From (1) and (2) we furthermore get

$$I \models_c \mathcal{C}_\beta(\ell^a)[\![Q]\!] \text{ and } \mathcal{H}(me)[\![Q]\!] \subseteq H$$

respectively, which in unison yields $\beta^{\ell^a_n}_{me}(Q) \sqsubseteq (I, H)$ by applying the induction hypothesis. Combining the above, we get

$$(\ell^a, \ell^a_n) \in I \;\wedge\; (\ell^a_n, me(n)) \in H \;\wedge\; \beta^{\ell^a_n}_{me}(Q) \sqsubseteq (I, H)$$

and by consulting the definition of β we see that $\beta^{\ell^a}_{me}(n^{\ell^a_n}[Q]) \sqsubseteq (I, H)$ as required.

The remaining cases are similar, or trivial applications of the induction hypothesis. ∎

The next proposition relates $\models$ and $\mathcal{C}_{\models}$; the condition $I \subseteq I^{\top}_{\star}$ is needed since the constraints generated by $\mathcal{C}_{\models}$ only takes labels from $\mathbf{Lab}_{\star}$ into account.

Proposition 6.9 *For all I we have*

$$\textit{if } I \subseteq I^{\top}_{\star} \textit{ then } (\forall H : (I, H) \models_{me_\star} P_\star \iff I \models_c \mathcal{C}_{\models}(me_\star, H)[\![P_\star]\!])$$

Proof. We first prove the "$\Rightarrow$"-direction. We do this by structural induction on the subprocesses P of $P_\star$, with the following induction hypothesis:

$$\forall me, H : ((I, H) \models_{me} P \;\Rightarrow\; I \models_c \mathcal{C}_{\models}(me, H)[\![P]\!])$$

Case (in ($P = in^{\ell^t} n.Q$)): Let me and H be given such that $(I,H) \models_{me} in^{\ell^t} n.Q$. By the definition of $\models$ we have

$$\begin{array}{ll}(1) & (I,H) \models_{me} Q \\ (2) & \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textsf{Lab}^a : (((\ell^a, \ell^t) \in I \ \wedge \ (\ell^{a''}, \ell^a) \in I \ \wedge \\ & \quad (\ell^{a''}, \ell^{a'}) \in I \ \wedge \ (\ell^{a'}, me(n)) \in H) \ \Rightarrow \ (\ell^{a''}, \ell^a) \in I)\end{array}$$

Using the induction hypothesis on (1) and the definition of $\models_c$ on (2) we obtain $I \models_c \mathcal{C}_{\models}(me, H)[\![Q]\!]$ and

$$I \models_c \Big\{ \{l^t\} \subseteq \texttt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \texttt{I}(l^{a'}) \texttt{ => } \{l^{a'}\} \subseteq \texttt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \texttt{I}(l^{a'}) \Big| \; \ell^a, \ell^{a''} \in \textsf{Lab}^a_\star, (\ell^{a'}, me(n)) \in H \Big\}$$

which gives us

$$I \models_c \mathcal{C}_{\models}(me, H)[\![in^{\ell^t} n.Q]\!]$$

The remaining cases are similar, or trivial applications of the induction hypothesis.

The "$\Leftarrow$"-direction is proved by structural induction on subprocesses P of $P_\star$, with the induction hypothesis:

$$\forall me, H : I \models_c \mathcal{C}_{\models}(me, H)[\![P]\!] \ \Rightarrow \ (I,H) \models_{me} P$$

Case (in ($P = in^{\ell^t} n.Q$)): Let me and H be given such that I is a solution to the constraints generated by $\mathcal{C}_{\models}$, i.e. $I \models_c \mathcal{C}_{\models}(me, H)[\![in^{\ell^t} n.Q]\!]$. By the definition of $\models_c$ we have

$$\begin{array}{ll}(1) & I \models_c \mathcal{C}_{\models}(me, H)[\![Q]\!] \\ (2) & I \models_c \Big\{ \{l^t\} \subseteq \texttt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \texttt{I}(l^{a'}) \texttt{ => } \{l^{a'}\} \subseteq \texttt{I}(l^{a''}) \texttt{ => } \\ & \quad \{l^a\} \subseteq \texttt{I}(l^{a'}) \Big| \; \ell^a, \ell^{a''} \in \textsf{Lab}^a_\star, (\ell^{a'}, me(n)) \in H \Big\}\end{array}$$

Using the induction hypothesis on (1) we get $(I,H) \models_{me} Q$. By the definition of $\models_c$ and the fact that $I \subseteq I_\star^\top$, we have

$$\forall \ell^a, \ell^{a'}, \ell^{a''} \in \textsf{Lab}^a : (((\ell^a, \ell^t) \in I \ \wedge \ (\ell^{a''}, \ell^a) \in I \ \wedge \ (\ell^{a''}, \ell^{a'}) \in I \ \wedge (\ell^{a'}, me(n)) \in H) \ \Rightarrow \ (\ell^{a''}, \ell^a) \in I)$$

In total we have $(I,H) \models_{me} in^{\ell^t} n.Q$.

The remaining cases are similar, or trivial applications of the induction hypothesis. ■

From Propositions 6.8 and 6.9 we have obtained, what we set out to do in this section. We have shown that if $\mathcal{H}(me_\star)[\![P_\star]\!] \subseteq H \subseteq H_\star^\top$, then we can find the I-component of the least analysis to $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge \ (I,H) \models_{me_\star} P_\star$ as the smallest solution to the set of constraints $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_{\models}(me_\star, H)[\![P_\star]\!]$. In conclusion we have that the least analysis is $(I, \mathcal{H}(me_\star)[\![P_\star]\!])$ where I is the least solution to the set of constraints $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_{\models}(me_\star, \mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!]$. In the following we will consider efficient ways to obtain solutions to sets of constraints.

6.4 Graph Algorithm

We will use a graph formulation of the constraints in order to compute the least solution to the set $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_\models(me_\star, H)[\![P_\star]\!]$.

In the following C will refer to the constraints to be solved. We define the set of all ambient labels occurring in a constraint, Lab^a_C: if l^a occurs in a $cc \in C$ then $\ell^a \in \mathsf{Lab}^a_C$. The nodes of the graph are $\mathtt{I}(l^a)$ for $\ell^a \in \mathsf{Lab}^a_C$. Each node p will have an associated data field D[p] and an edge field E[p]. The algorithm treats a constraints depending on which of the two following forms the constraint is in.

$$\{l^{at}\} \subseteq \mathtt{I}(l^a) \tag{6.1}$$

$$\{l^{at}_1\} \subseteq \mathtt{I}(l^a_1) \texttt{ => } \cdots \texttt{ => } \{l^{at}_n\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^a), n \geq 1 \tag{6.2}$$

The constraints in form (6.1) are used to give initial values to the D fields:

$$\mathrm{D}[p] = \left\{\ell^{at} \,\middle|\, (\{l^{at}\} \subseteq p) \in C\right\}$$

The graph will have edges for each constraint in form (6.2). For a constraint

$$\{l^{at}_1\} \subseteq p_1 \texttt{ => } \cdots \texttt{ => } \{l^{at}_n\} \subseteq p_n \texttt{ => } \{l^{at}\} \subseteq p, n \geq 1$$

edges from p_i to p are constructed for $i \in \{1, \ldots, n\}$. Each edge will be decorated with the constraint.

Having constructed the graph we now traverse the edges in order to ensure that the constraints are met by extending the data fields. Each time a D[p_i] is extended with a label not previously there (and this incorporates the situation where D[p_i] is initially set to a non-empty set) all its outgoing edges are considered. An edge decorated with $\{l^{at}_1\} \subseteq p_1 \texttt{ => } \cdots \texttt{ => } \{l^{at}_n\} \subseteq p_n \texttt{ => } \{l^{at}\} \subseteq p$ is only traversed (and ℓ^{at} added to D[p]) if in fact $\ell^{at}_1 \in \mathrm{D}[p_1], \ldots, \ell^{at}_{n-1} \in \mathrm{D}[p_{n-1}]$.

To be more specific consider the algorithm of Table 6.5. It takes as input a set C of constraints and produces as output a solution I. It operates on the main data structures:

- a *worklist* W that is a list of nodes whose outgoing edges should be traversed;
- a *data array* D that for each node gives an element of $\mathcal{P}(\mathsf{Lab})$; and
- an *edge array* E that for each node gives a list of constraints from which a list of the successor nodes can be computed.

The set Nodes consists of nodes $\mathtt{I}(l^a)$ for all $\ell^a \in \mathsf{Lab}^a_C$. The first step of the algorithm is to 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 $\mathtt{add}(q, d)$ that incorporates d into D[q] and adds q to the worklist

INPUT: Set of constraints C

OUTPUT: I

METHOD: Step 1: Initialisation
W := `nil`;
for q in Nodes do D[q] := $\emptyset$;
for q in Nodes do E[q] := `nil`;

Step 2: Building the graph
for cc in C do
case cc of
$\{l^{at}\} \subseteq p$: `add`(p, $\{\ell^{at}\}$);
$\{l_1^{at}\} \subseteq p_1$ => $\cdots$ => $\{l_n^{at}\} \subseteq p_n$ => $\{l^{at}\} \subseteq p$, $(n \geq 1)$:
E[p_1] := `cons`(cc,E[p_1]);
$\vdots$
E[p_n] := `cons`(cc,E[p_n]);

Step 3: Iteration
while W$\neq$ `nil` do
q:= `head`(W); W:= `tail`(W);
for ($\{l_1^{at}\} \subseteq p_1$ => $\cdots$ => $\{l_n^{at}\} \subseteq p_n$ => $\{l^{at}\} \subseteq p$) in E[q] do
if $\{\ell_1^{at}\} \subseteq$ D[p_1] and $\cdots$ and $\{\ell_n^{at}\} \subseteq$ D[p_n]
then `add`(p,$\{\ell^{at}\}$);

Step 4: Recording the solution
$I = \left\{(\ell^a, \ell^{a'}) \,\middle|\, \ell^a \in \textbf{Lab}^a_C, \ell^{a'} \in \mathrm{D}[\mathtt{I}(l^a)]\right\}$

USING: procedure `add`(q,d) is
if $\neg(d \subseteq \mathrm{D}[q])$
then D[q] := D[q] $\cup\, d$;
W:= `cons`(q,W);

Table 6.5: Graph Algorithm to solve constraints

if d was not part of D[q]. The third step is to continue propagating contributions along edges as long as the worklist is non-empty. The fourth and final step is to record the solution in a more familiar form.

The following proposition shows that the algorithm computes the solution we want.

Proposition 6.10 *Given finite input C the algorithm terminates and the result*

I produced by the algorithm satisfies

$$I = \bigcap \{I' \mid I' \models_c C\}$$

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 $\mathrm{D}[q]$ never decrease and that they can be increased at most a finite number of times. It is also immediate that a node q is added to the worklist only if some value of $\mathrm{D}[q]$ actually increased. For each node placed on the worklist 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 worklist. This guarantees termination. Next let I' be a solution to $I' \models_c C$. It is possible to show that the following invariant

$$\forall \ell^a \in \mathsf{Lab}^a_C : \left\{ (\ell^a, \ell^{a'}) \,\middle|\, \ell^{a'} \in \mathrm{D}[\mathtt{I}(l^a)] \right\} \subseteq I'$$

is maintained at all points after step 1. It follows that $I \subseteq I'$ upon completion of the algorithm. We prove that $I \models_c C$ by contradiction. So suppose there exists $cc \in C$ such that $I \models_c cc$ does not hold. If cc is $\{l^{at}\} \subseteq p$ then step 2 ensures that $\{\ell^{at}\} \subseteq \mathrm{D}[p]$ and this is maintained throughout the algorithm; hence cc cannot have this form. If cc is $\{l^{at}_1\} \subseteq p_1$ `=>` $\cdots$ `=>` $\{l^{at}_n\} \subseteq p_n$ `=>` $\{l^{at}\} \subseteq p, (n \geq 1)$ it must be the case that $\mathrm{D}[p_i] \neq \emptyset$ for $i \in \{1, \ldots, n\}$ since otherwise $I \models_c cc$ would hold; now consider the last time one of the $\mathrm{D}[p_i]$ was modified and note that p_i was placed on the worklist at that time (by procedure `add`); since the final worklist is empty we must have considered the constraint cc (which is in $\mathrm{E}[p_i]$) and updated $\mathrm{D}[p]$ accordingly; hence cc cannot have this form. Thus $I \models_c cc$ for all $cc \in C$. We have now shown that $I \models_c C$ and that $I \subseteq I'$ whenever $I' \models_c C$. It now follows that

$$I = \bigcap \{I' \mid I' \models_c C\}$$

as required. ■

Table 6.6: Initialisation of data structure for example process

p	$\mathrm{D}[p]$	E[p]
$\mathtt{I}(l^a_\star)$	$\{\ell^a_n, \ell^a_m\}$	$[\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^t_2\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^t_2\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^t_2\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^t_1\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^t_1\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^t_1\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{\ell^a_m\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{\ell^a_m\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{\ell^a_m\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{\ell^a_m\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star),$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_n) \texttt{ => } \{l^a_n\} \subseteq \mathtt{I}(l^a_\star),$

continued on next page

continued from previous page

p	D[p]	E[p]
		$\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_1^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$]
$\mathtt{I}(l_n^a)$	$\{\ell_1^t\}$	[$\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_2^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_1^t\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_2^t\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(l_\star^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_\star^a\} \subseteq \mathtt{I}(l_\star^a)$, $\{l_1^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(l_n^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$, $\{l_1^t\} \subseteq \mathtt{I}(l_n^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{\ell_m^a\} \subseteq \mathtt{I}(\ell_m^a)$ => $\{l_n^a\} \subseteq \mathtt{I}(\ell_m^a)$,

continued on next page

continued from previous page

p	D[p]	E[p]
		$\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(\ell^a_m)$]
$\mathtt{I}(l^a_m)$	$\{\ell^t_2\}$	[$\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_2\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^t_1\} \subseteq \mathtt{I}(l^a_\star)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_\star)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$, $\{l^t_2\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_\star)$, $\{l^t_1\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(\ell^a_m)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_n\} \subseteq \mathtt{I}(\ell^a_m)$, $\{l^t_1\} \subseteq \mathtt{I}(l^a_\star)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ $\mathtt{=>}$ $\{l^a_\star\} \subseteq \mathtt{I}(\ell^a_m)$,

continued on next page

W	$[\mathtt{I}(l^a_m), \mathtt{I}(l^a_\star), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$	$[\mathtt{I}(l^a_\star), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$	$[\mathtt{I}(l^a_m), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$
p	D[p]	D[p]	D[p]
$\mathtt{I}(l^a_\star)$	$\{\ell^a_n, \ell^a_m\}$	$\{\ell^a_n, \ell^a_m\}$	$\{\ell^a_n, \ell^a_m\}$
$\mathtt{I}(l^a_n)$	$\{\ell^t_1\}$	$\{\ell^t_1\}$	$\{\ell^t_1\}$
$\mathtt{I}(l^a_m)$	$\{\ell^t_2\}$	$\{\ell^t_2\}$	$\{\ell^a_n, \ell^t_2\}$

W	$[\mathtt{I}(l^a_m), \mathtt{I}(l^a_m), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$	$[\mathtt{I}(l^a_m), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$	$[\mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$
p	D[p]	D[p]	D[p]
$\mathtt{I}(l^a_\star)$	$\{\ell^a_n, \ell^a_m\}$	$\{\ell^a_n, \ell^a_m\}$	$\{\ell^a_n, \ell^a_m\}$
$\mathtt{I}(l^a_n)$	$\{\ell^t_1\}$	$\{\ell^t_1\}$	$\{\ell^t_1\}$
$\mathtt{I}(l^a_m)$	$\{\ell^a_n, \ell^a_m, \ell^t_1, \ell^t_2\}$	$\{\ell^a_n, \ell^a_m, \ell^t_1, \ell^t_2\}$	$\{\ell^a_n, \ell^a_m, \ell^t_1, \ell^t_2\}$

W	$[\mathtt{I}(l^a_\star)]$	[ ]
p	D[p]	D[p]
$\mathtt{I}(l^a_\star)$	$\{\ell^a_n, \ell^a_m\}$	$\{\ell^a_n, \ell^a_m\}$
$\mathtt{I}(l^a_n)$	$\{\ell^t_1\}$	$\{\ell^t_1\}$
$\mathtt{I}(l^a_m)$	$\{\ell^a_n, \ell^a_m, \ell^t_1, \ell^t_2\}$	$\{\ell^a_n, \ell^a_m, \ell^t_1, \ell^t_2\}$

Table 6.7: Iteration steps of example process.

continued from previous page

p	D[p]	E[p]
		$[\{l^t_1\} \subseteq \mathtt{I}(l^a_\star)$ `=>` $\{l^a_\star\} \subseteq \mathtt{I}(l^a_n)$ `=>` $\{\ell^a_m\} \subseteq \mathtt{I}(l^a_n)$ `=>` $\{l^a_\star\} \subseteq \mathtt{I}(\ell^a_m)]$

Example 6.11 Let us consider how the algorithm operates on the constraints generated from Example 6.7. After step 2 the data structure W has been initialised to

$$W = [\mathtt{I}(l^a_m), \mathtt{I}(l^a_\star), \mathtt{I}(l^a_n), \mathtt{I}(l^a_\star)]$$

and the data structures D and E have been initialised as in Table 6.6. The algorithm will now iterate through the worklist and update the data structures W and D as described by step 3. The various intermediate stages are recorded in Table 6.7. The algorithm computes the solution for I in the last "box":

$$I = \left\{(\ell^a_\star, \ell^a_n), (\ell^a_\star, \ell^a_m), (\ell^a_n, \ell^t_1), (\ell^a_m, \ell^a_n), (\ell^a_m, \ell^a_m), (\ell^a_m, \ell^t_1), (\ell^a_m, \ell^t_2)\right\}$$

We have that $(I, \mathcal{H}(me_\star)[\![P_\star]\!])$ is the least analysis to $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I, H) \ \wedge$ $(I, H) \models_{me_\star} P_\star$, where $P_\star$ and $me_\star$ were given in Example 6.7, and $\mathcal{H}(me_\star)[\![P_\star]\!] =$ $\{(\ell^a_m, \mu), (\ell^a_n, \mu_n)\}$ as presented in Example 6.3. $\diamond$

6.4.1 Considering time bounds

In this section it is examined how much time is needed to solve a set of constraints generated by the functions $\mathcal{C}_\beta$ and $\mathcal{C}_\models$. It is important to notice that the lengths of these are independent of the process.

The proof showing that the algorithm terminates can be refined to show that it takes at most $O(t \cdot u)$ steps, if the number of constraints is t and the size of the set of occurring constants in the constraints is u. This means that each data field can be enlarged at most $O(u)$ times. Assuming that the operations upon $\mathrm{D}[p]$ take unit time we can perform the following calculations: step 1 takes time $O(u)$, step 2 takes time $O(t)$, and step 4 takes time $O(u)$; step 3 traverses each of the t edges at most $O(u)$ times and hence takes time $O(t \cdot u)$; it follows that the overall algorithm takes no more than $O(t \cdot u)$ basic steps.

Recall that the constraints to be solved in order to find the least analysis are $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_\models(me_\star, \mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!]$. If the process $P_\star$ has size s there are $O(s)$ constraints in the form (6.1):

$$\{l^{at}\} \subseteq \mathtt{I}(l^a)$$

and $O(s^4)$ constraints in the form (6.2):

$$\{l_1^{at}\} \subseteq \mathtt{I}(l_1^a) \texttt{ => } \{l_2^{at}\} \subseteq \mathtt{I}(l_2^a) \texttt{ => } \{l_3^{at}\} \subseteq \mathtt{I}(l_3^a) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^a)$$

To see this note that $\mathcal{C}_\beta$ generates $O(1)$ constraints of the first form for each of the $O(s)$ constituents of $P_\star$ whereas $\mathcal{C}_\models$ generates $O(s^3)$ constraints of the second form for each constituent of $P_\star$ (we use $O(s)$ as a bound on the size of $\{\ell^a \mid (\ell^a, \mu) \in \mathcal{H}(me_\star)[\![P_\star]\!]\}$). Now we have that $t = O(s^4)$ and $u = |\mathbf{Lab}_\star| = O(s)$ so the least solution can be computed in time $O(s^5)$.

Remark 6.12 If we had chosen to represent $(\ell^a, \ell^{at}) \in I$ as $(l^a, l^{at}) \in \mathtt{I}$ as discussed in Remark 6.6 and changed the algorithm accordingly, the size of the set of occurring constants would have been $O(s^2)$ (the cardinality of $\mathbf{Lab}^a_\star \times \mathbf{Lab}_\star$); this means that u above would be $O(s^2)$, the number of constraints t would still be $O(s^4)$, so we can only put the bound of $O(s^6)$ on the worst case time complexity. $\diamond$

6.5 Improving the time bound

For a process of size s, we have generated $O(s^4)$ constraints which can be solved in $O(s^5)$ time. An obvious way to improve this is to find a way to generate fewer constraints. For the rest of the chapter we shall focus on the constraints generated by $\mathcal{C}_\models$ since $\mathcal{C}_\beta$ only accounts for $O(s)$ constraints. By imposing simple restrictions on the capabilities in the process it can be shown that only $O(s^3)$ constraints are ever generated. We will show how to use this to compute the least solution in time $O(s^4)$. Subsequently, we will show how, by generating slightly different constraints, we can reduce the number of generated constraints to $O(s^3)$ in the general case where there is no restrictions on the process. Again the least solution can be computed in $O(s^4)$ time.

First of all we would like to see if there exists processes that really do produce $\Theta(s^4)$ constraints with the current generating functions, or if we could improve the bound given additional insight. The following example shows that we cannot improve the bound in general.

Example 6.13 Consider a process P of size s and assume that it has $\Theta(s)$ occurrences of ambients with a name n bound at a restriction with marker μ and that the ambients all have distinct ambient labels. Assume furthermore that P has $\Theta(s)$ occurrences of in-capabilities where the ambient name of the capability is bound at a restriction with marker μ and that all thread labels are distinct. An example of a such process is

$$(\nu_\mu n)(n^{\ell^a_{n_1}}[\mathbf{0}] \mid \cdots \mid n^{\ell^a_{n_t}}[\mathbf{0}] \mid in^{\ell^t_1} n \mid \cdots \mid in^{\ell^t_u} n)$$

The size of $\{\ell^a \mid (\ell^a, \mu) \in \mathcal{H}(me)[\![P]\!]\}$ is then $\Theta(s)$ for all me. Each of the $\Theta(s)$ occurrences of an in-capability will then generate $\Theta(s^3)$ constraints for a total of $\Theta(s^4)$ constraints. $\diamond$

6.5.1 Restrictions on the capabilities

In this section we will show that by imposing simple restrictions on the capabilities in the process $P_\star$ only $O(s^3)$ constraints are generated. In the calculation of the number of generated constraints the size of $\{\ell^a \mid (\ell^a, \mu) \in \mathcal{H}(me_\star)[\![P_\star]\!]\}$ is estimated at $O(s)$ for all μ. An important thing to notice, however, is that the size of $\mathcal{H}(me_\star)[\![P_\star]\!]$ is limited by the size of $P_\star$. This is expressed by the following fact.

Fact 6.14 *Assume that the process $P_\star$ has size s. Then $|\mathcal{H}(me_\star)[\![P_\star]\!]|$ is $O(s)$.*

Proof. From Table 6.1 we see that for each of the $O(s)$ ambients in $P_\star$ at most one element is added to $\mathcal{H}(me_\star)[\![P_\star]\!]$. ■

Please recall the set of stable capabilities introduced in Chapter 5:

$$\textbf{CAPA} = \left\{ \mathsf{in}^{\ell^t} \mu, \mathsf{out}^{\ell^t} \mu, \mathsf{open}^{\ell^t} \mu \,\middle|\, \ell^t \in \textbf{Lab}^t \wedge \mu \in \textbf{Mar} \right\}$$

We impose a restriction on the number of capabilities of each kind in $P_\star$ with respect to $me_\star$: for each marker, $\mu \in \textbf{Mar}$ at most a constant number, k, of each of the stable capabilities $\mathsf{in}^{\ell^t} \mu$, $\mathsf{out}^{\ell^t} \mu$ and $\mathsf{open}^{\ell^t} \mu$ are allowed in $P_\star$ with respect to $me_\star$. Consider the constraints generated by $\mathcal{C}_{\models}(me_\star, \mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!]$. We can put a bound on the number of generated constraints by:

$$3 \cdot \sum_{\mu \in \textbf{Mar}} k \cdot O(s^2)\, |\{\ell^a \mid (\ell^a, \mu) \in \mathcal{H}(me_\star)[\![P_\star]\!]\}| \in O(s^3)$$

We see that with this restriction only $O(s^3)$ constraints are generated in total, which we can solve in time $O(s^4)$ by the algorithm.

Here we have considered imposing restrictions on the given process and marker environment to reduce the number of constraints, but as shown in Example 6.13 we cannot improve the bound in the general case. In the following section we will improve the bound in the general case by generating a different set of constraints.

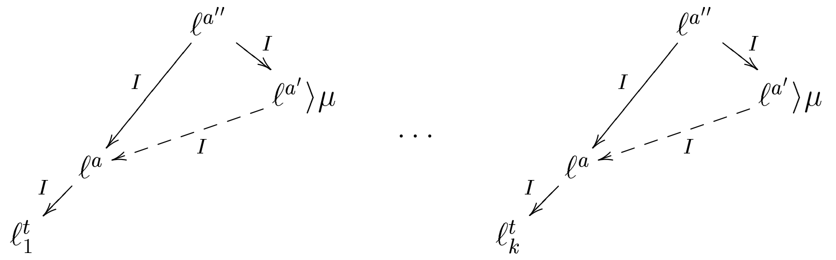
6.5.2 Improving the time bound in general

We will now prove that the time bound can be improved to $O(s^4)$ in the general case. We consider the set of generated constraints, where we focus on the similarities of the constraints. We end up with a new way of generating constraints where we combine the similarities of the constraints and only generate $O(s^3)$ constraints which can be solved in time $O(s^4)$.

In the following the intuition behind the changes of constraints is explained; we elaborate on the idea behind the restriction in the previous section. We introduce the set of stable capabilities without labels

$$\mathsf{MCap} = \left\{ in\,\mu, out\,\mu, open\,\mu \,\middle|\, \mathsf{in}^{\ell^t}\mu, \mathsf{out}^{\ell^t}\mu, \mathsf{open}^{\ell^t}\mu \in \mathsf{SCap} \right\}$$

We will use $in\,\mu \in \mathsf{MCap}$ in order to illustrate how the constraints are combined. Let us assume that the capabilities corresponding to $in\,\mu$ occurs in the process $P_\star$ with labels $\ell^t_1, \ldots, \ell^t_k$, i.e. $\left\{ \ell^t \,\middle|\, \mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me_\star}(P_\star) \right\} = \{\ell^t_1, \ldots, \ell^t_k\}$. We will illustrate the checks that should hold with respect to these capabilities in order for a proposed analysis (I, H) to be acceptable by the diagrams from Chapter 4:



Written as constraints this amounts to:

$$\{l^t_1\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})$$
$$\vdots$$
$$\{l^t_k\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})$$

where $\ell^a, \ell^{a''} \in \mathsf{Lab}^a_\star$ and $(\ell^{a'}, \mu) \in H$.

The main insight behind the change of constraints is the amount of redundancy in these constraints. We introduce a much simplified version of these constraints by introducing the auxiliary functions f_{l^a} and g_{l^a}, which are only used to illustrate the similarities of the constraints.

$$\begin{aligned} f_{l^a}(l^t) &= \{l^t\} \subseteq \mathtt{I}(l^a) \\ g_{l^a}(l^{a'}, l^{a''}) &= \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'}) \end{aligned}$$

For each $\ell^a \in \mathbf{Lab}^a_\star$ we collect the constraints:

$$\begin{gathered} f_{l^a}(l^t_1) \texttt{ => } g_{l^a}(l^{a'}, l^{a''}) \\ \vdots \\ f_{l^a}(l^t_k) \texttt{ => } g_{l^a}(l^{a'}, l^{a''}) \end{gathered}$$

for $(\ell^{a'}, \mu) \in H$ and $\ell^{a''} \in \mathbf{Lab}^a_\star$. It is now obvious that the right hand sides of the constraints are identical for a given ℓ^a, and in order to reduce the number of constraints we introduce a fresh variable $\mathtt{F}(in\,\mu)$ and rewrite the above system of constraints as

$$\begin{gathered} f_{l^a}(l^t_1) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(in\,\mu) \\ \vdots \\ f_{l^a}(l^t_k) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(in\,\mu) \\ \{l^a\} \subseteq \mathtt{F}(in\,\mu) \texttt{ => } g_{l^a}(l^{a'}, l^{a''}) \end{gathered}$$

for $(\ell^{a'}, \mu) \in H$ and $\ell^{a''} \in \mathbf{Lab}^a_\star$. The intuitive meaning of the fresh variable $\mathtt{F}(in\,\mu)$ is that if one of $f_{l^a}(l^t_i), 1 \leq i \leq k$, holds then l^a will be an element in $\mathtt{F}(in\,\mu)$. Since the expression

$$f_{l^a}(l^t_1) \texttt{ => } g_{l^a}(l^{a'}, l^{a''})$$

represents a set of constraints with a constraints for each combination of $\ell^{a'}$ and $\ell^{a''}$ where $(\ell^{a'}, \mu) \in \mathcal{H}$ and $\ell^{a''} \in \mathbf{Lab}^a_\star$, the size of the original set of constraints is

$$k \cdot |\mathbf{Lab}^a_\star| \cdot |\{\ell^a \mid (\ell^a, \mu) \in H\}|$$

In contrast to this, the size of the transformed set of constraints will only be

$$k + |\mathbf{Lab}^a_\star| \cdot |\{\ell^a \mid (\ell^a, \mu) \in H\}|$$

since the dependency of $\ell^{a'}$ and $\ell^{a''}$ has been removed from the k first expressions.

With this characterisation of the constraints for the capability $in\,\mu$, it is straightforward to find the analogous constraints for the capabilities $out\,\mu$ and $open\,\mu$.

The generating function $\mathcal{C}'_{\models}$ for the collected constraints is shown in Table 6.8. Having introduced $\mathcal{C}'_{\models}$ we proceed by proving its relationship to the closure property, $\models$. There are two natural ways to obtain this relationship. The first possibility is to show a relationship between $\mathcal{C}_{\models}$ and $\mathcal{C}'_{\models}$ and use the already proved

$$
\begin{aligned}
&\mathcal{C}'_{\models}(me, H)[\![P]\!] = \\
&\quad \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{F}(in\,\mu) \,\Big|\, \ell^a \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P) \Big\} \\
&\quad \cup \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{F}(out\,\mu) \,\Big|\, \ell^a \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{out}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P) \Big\} \\
&\quad \cup \Big\{ \{l^t\} \subseteq \mathtt{I}(l^a) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{F}(open\,\mu) \,\Big|\, \ell^a \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{open}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P) \Big\} \\
&\quad \cup \Big\{ \{l^a\} \subseteq \mathtt{F}(in\,\mu) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{I}(l^{a''}) \ \texttt{=>}\ \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{I}(l^{a'}) \,\Big| \\
&\qquad\quad \ell^a, \ell^{a''} \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P), (\ell^{a'}, \mu) \in H \Big\} \\
&\quad \cup \Big\{ \{l^a\} \subseteq \mathtt{F}(out\,\mu) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{I}(l^{a'}) \ \texttt{=>}\ \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \ \texttt{=>}\ \{l^a\} \subseteq \mathtt{I}(l^{a''}) \,\Big| \\
&\qquad\quad \ell^a, \ell^{a''} \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{out}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P), (\ell^{a'}, \mu) \in H \Big\} \\
&\quad \cup \Big\{ \{l^a\} \subseteq \mathtt{F}(open\,\mu) \ \texttt{=>}\ \{l^{a'}\} \subseteq \mathtt{I}(l^a) \ \texttt{=>}\ \{l^{at}\} \subseteq \mathtt{I}(l^{a'}) \ \texttt{=>}\ \{l^{at}\} \subseteq \mathtt{I}(l^a) \,\Big| \\
&\qquad\quad \ell^a \in \mathsf{\textbf{Lab}}^a_\star, \mathsf{open}^{\ell^t}\mu \in \mathrm{CAPA}_{me}(P), (\ell^{a'}, \mu) \in H, l^{at} \in \mathsf{\textbf{Lab}}_\star \Big\}
\end{aligned}
$$

Table 6.8: Modified Constraints (Closure Property)

relationship between $\mathcal{C}_{\models}$ and $\models$. The second possibility is to prove the relationship directly. Since our interest in this thesis lies in the closure property rather than the different constraints systems, we have adopted the second possibility, which allows us to introduce an alternative formulation of the closure property (Proposition 6.15). We depict the relationships between $\mathcal{C}_{\models}$, $\mathcal{C}'_{\models}$ and $\models$ by the following "commutative" diagram:

$$
\begin{tikzcd}
 & \models \arrow[dl, dash] \arrow[dr, dash] & \\
\mathcal{C}_{\models} \arrow[rr, dash, dashed] & & \mathcal{C}'_{\models}
\end{tikzcd}
$$

where the solid lines represents the relationships that are shown directly in the thesis. After having obtained the relationship, we go on to show that we can obtain the least analysis from the smallest solution to constraints generated by $\mathcal{C}_\beta$ and $\mathcal{C}'_{\models}$.

In the generation of constraints by $\mathcal{C}'_{\models}$, new variables are introduced; this is incorporated by extending the **var**-production in the grammar for constraints:

$$\textbf{var} \quad ::= \quad \mathtt{I}(l^a) \mid \mathtt{F}(in\,\mu) \mid \mathtt{F}(out\,\mu) \mid \mathtt{F}(open\,\mu), \ell^a \in \mathsf{\textbf{Lab}}^a, \mu \in \mathsf{\textbf{Mar}}$$

Solutions are now pairs (I, F) where $F \subseteq \mathsf{\textbf{MCap}} \times \mathsf{\textbf{Lab}}^a$ and the evaluation of variables is extended:

$$
\begin{aligned}
(I, F)[\![\mathtt{F}(in\,\mu)]\!] &= \{\ell^a \mid (in\,\mu, \ell^a) \in F\} \\
(I, F)[\![\mathtt{F}(out\,\mu)]\!] &= \{\ell^a \mid (out\,\mu, \ell^a) \in F\} \\
(I, F)[\![\mathtt{F}(open\,\mu)]\!] &= \{\ell^a \mid (open\,\mu, \ell^a) \in F\}
\end{aligned}
$$

(res) $\quad \mathcal{F}(me, (\nu_\mu n)P, I) = \mathcal{F}(me[n \mapsto \mu], P, I)$

$(zero)$ $\quad \mathcal{F}(me, \mathbf{0}, I) = \emptyset$

(par) $\quad \mathcal{F}(me, P \mid Q, I) = \mathcal{F}(me, P, I) \cup \mathcal{F}(me, Q, I)$

$(repl)$ $\quad \mathcal{F}(me, !P, I) = \mathcal{F}(me, P, I)$

(amb) $\quad \mathcal{F}(me, n^{\ell^a}[P], I) = \mathcal{F}(me, P, I)$

(in) $\quad \mathcal{F}(me, in^{\ell^t} n.P, I) = \mathcal{F}(me, P, I) \cup \{(in\,\mu, \ell^a) \mid (\ell^a, \ell^t) \in I\}$, $\quad$ *where* $me(n) = \mu$

(out) $\quad \mathcal{F}(me, out^{\ell^t} n.P, I) = \mathcal{F}(me, P, I) \cup \{(out\,\mu, \ell^a) \mid (\ell^a, \ell^t) \in I\}$, $\quad$ *where* $me(n) = \mu$

$(open)$ $\quad \mathcal{F}(me, open^{\ell^t} n.P, I) = \mathcal{F}(me, P, I) \cup \{(open\,\mu, \ell^a) \mid (\ell^a, \ell^t) \in I\}$, $\quad$ *where* $me(n) = \mu$

Table 6.9: The function $\mathcal{F}$

In order to prove the correspondence between $\mathcal{C}'_{\models}$ on one hand and $\models$ on the other, we introduce the function: $\mathcal{F} : \mathbf{\mathsf{MEnv}} \times \mathbf{\mathsf{Proc}} \times \mathbf{\mathsf{InAmb}} \to \mathcal{P}(\mathbf{\mathsf{MCap}} \times \mathbf{\mathsf{Lab}}^a)$, which is defined by

$$\begin{array}{rcl}
\mathcal{F}(me, P, I) & = & \left\{(in\,\mu, \ell^a) \,\middle|\, \mathsf{in}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \;\wedge\; (\ell^a, \ell^t) \in I\right\} \\
& \cup & \left\{(out\,\mu, \ell^a) \,\middle|\, \mathsf{out}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \;\wedge\; (\ell^a, \ell^t) \in I\right\} \\
& \cup & \left\{(open\,\mu, \ell^a) \,\middle|\, \mathsf{open}^{\ell^t} \mu \in \mathrm{CAPA}_{me}(P) \;\wedge\; (\ell^a, \ell^t) \in I\right\}
\end{array}$$

Intuitively, $\mathcal{F}$ can be understood as computing the “real” value of the variable F with respect to I; this will be made clear by Lemma 6.18. $\mathcal{F}$ can be calculated as shown in Table 6.9. As a step towards relating $\mathcal{C}'_{\models}$ and $\models$, we express $\models$ in terms of $\mathcal{F}$ by the following proposition.

Proposition 6.15 *For all I, H, me and P where $fn(P) \subseteq \mathrm{dom}(me)$*

$$(I,H) \models_{me} P \text{ iff } \forall \mu \in \textsf{Mar}:$$
$$\left(\begin{array}{l} \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textsf{Lab}^a : (((in\,\mu, \ell^a) \in \mathcal{F}(me,P,I) \;\wedge\; (\ell^{a''}, \ell^a) \in I \;\wedge \\ \quad (\ell^{a''}, \ell^{a'}) \in I \;\wedge\; (\ell^{a'}, \mu) \in H) \Rightarrow (\ell^{a'}, \ell^a) \in I) \;\wedge \\ \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textsf{Lab}^a : (((out\,\mu, \ell^a) \in \mathcal{F}(me,P,I) \;\wedge\; (\ell^{a'}, \ell^a) \in I \;\wedge \\ \quad (\ell^{a''}, \ell^{a'}) \in I \;\wedge\; (\ell^{a'}, \mu) \in H) \Rightarrow (\ell^{a''}, \ell^a) \in I) \;\wedge \\ \forall \ell^a, \ell^{a'} \in \textsf{Lab}^a : (((open\,\mu, \ell^a) \in \mathcal{F}(me,P,I) \;\wedge\; (\ell^a, \ell^{a'}) \in I \;\wedge \\ \quad (\ell^{a'}, \mu) \in H) \Rightarrow \left\{(\ell^a, \ell') \,\middle|\, (\ell^{a'}, \ell') \in I\right\} \subseteq I) \end{array}\right)$$

Proof. By structural induction on P. We abbreviate the right hand side above by $\mathrm{AP}(I,H,me,P)$. In the following we show the "$\Rightarrow$" direction; the "$\Leftarrow$" direction is analogous.

Case (in ($P = in^{\ell^t} n.Q$)): From the definition of $\models$ we have

$$\begin{array}{ll} (1) & (I,H) \models_{me} Q \\ (2) & \forall \ell^a, \ell^{a'}, \ell^{a''} \in \textsf{Lab}^a : ((\ell^a, \ell^t) \in I \;\wedge\; (\ell^{a''}, \ell^a) \in I \;\wedge\; (\ell^{a''}, \ell^{a'}) \in I \\ & \qquad \wedge\; (\ell^{a'}, me(n)) \in H) \Rightarrow (\ell^{a'}, \ell^a) \in I \end{array}$$

We apply the induction hypothesis to (1) to obtain $\mathrm{AP}(I,H,me,Q)$. Combining this with (2) we obtain $\mathrm{AP}(I,H,me,in^{\ell^t} n.Q)$ by calculations based on Table 6.9.

The remaining cases are similar, or trivial applications of the induction hypothesis. ■

Proposition 6.17 below shows that if (I,F) is a solution to constraints generated by $\mathcal{C}_\beta$ and $\mathcal{C}'_{\models}$ with respect to H then (I,H) is an acceptable analysis.

We will use the alternative formulation of the closure property from Proposition 6.15 in the proof of the following lemma.

Lemma 6.16 *For all I, H and F we have*

$$(I,F) \models_c \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!] \;\wedge\; I \subseteq I_\star^\top \Rightarrow (I,H) \models_{me_\star} P_\star$$

Proof. We first show that $\mathcal{F}(me_\star, P_\star, I) \subseteq F$ holds under the assumption that $(I,F) \models_c \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$ and $I \subseteq I_\star^\top$. Assume that $(c, \ell^a) \in \mathcal{F}(me_\star, P_\star, I)$ for some $c \in \textsf{MCap}$ and $\ell^a \in \textsf{Lab}^a$.

Case ($c = in\,\mu$): By definition of $\mathcal{F}$ we have that $\mathsf{in}^{\ell^t} \mu \in \mathrm{CAPA}_{me_\star}(P_\star)$ and $(\ell^a, \ell^t) \in I$ for some $\ell^t \in \textsf{Lab}^t$. Since $I \subseteq I_\star^\top$ it follows that $\ell^a \in \textsf{Lab}^a_\star$ so $\{\{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(in\,\mu)\} \subseteq \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$. From $(I,F) \models_c \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$, it follows that $(in\,\mu, \ell^a) \in F$.

The cases for $c = out\,\mu$ and $c = open\,\mu$ are similar.

We show that $(I, H) \models_{me_\star} P_\star$ by the alternative formulation of $\models$ from Proposition 6.15. Assume that $\ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a$ and $\mu \in \mathbf{Mar}$ are given such that $(in\,\mu, \ell^a) \in \mathcal{F}(me_\star, P_\star, I), (\ell^{a''}, \ell^a) \in I, (\ell^{a''}, \ell^{a'}) \in I$, and $(\ell^{a'}, \mu) \in H$ holds. By the definition of $\mathcal{F}$ it follows that there exists ℓ^t such that $\mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me_\star}(P_\star)$. From $I \subseteq I_\star^\top$ we have $\ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}_\star^a$, so

$$\Big\{\{l^a\} \subseteq \mathtt{F}(in\,\mu) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})\Big\} \subseteq \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$$

Since $\mathcal{F}(me_\star, P_\star, I) \subseteq F$ we have $(in\,\mu, \ell^a) \in F$ and by $(I, F) \models_c \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$ it follows that $(\ell^{a'}, \ell^a) \in I$. The cases for $out\,\mu$ and $open\,\mu$ are similar. ∎

Proposition 6.17 *For all I, H and F we have*

$$(I, F) \models_c (\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]) \ \wedge\ I \subseteq I_\star^\top \ \wedge\ H \subseteq \mathcal{H}(me_\star)[\![P_\star]\!] \ \Rightarrow \ \beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I, H) \ \wedge\ (I, H) \models_{me_\star} P_\star$$

Proof. The first conjunction follows by an argument analogous to the proof of Proposition 6.8. The second conjunction follows from Lemma 6.16. ∎

We will now show by Proposition 6.19 that if (I, H) is an acceptable analysis then $(I,\mathcal{F}(me_\star, P_\star, I))$ is a solution to the corresponding constraints generated by $\mathcal{C}_\beta$ and $\mathcal{C}'_{\models}$.

Lemma 6.18 *For all I and H we have*

$$(I, H) \models_{me_\star} P_\star \Rightarrow (I, \mathcal{F}(me_\star, P_\star, I)) \models_c \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$$

Proof. By cases on $cc \in \mathcal{C}'_{\models}(me_\star, H)[\![P_\star]\!]$. First, we consider the case

$$cc = \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(in\,\mu)$$

where $\mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me_\star}(P_\star)$ and $\ell^a \in \mathbf{Lab}_\star$. Assume that $(\ell^a, \ell^t) \in I$; by definition of $\mathcal{F}$ we have that $(in\,\mu, \ell^a) \in \mathcal{F}(me_\star, P_\star, I)$, so $(I, \mathcal{F}(me_\star, P_\star, I)) \models_c cc$. The cases $cc = \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(out\,\mu)$ and $cc = \{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(open\,\mu)$ are similar.

We will now consider the case

$$cc = \{l^a\} \subseteq \mathtt{F}(in\,\mu) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})$$

where $\ell^a, \ell^{a''} \in \mathbf{Lab}_\star^a$, $(\ell^{a'}, \mu) \in H$ and $\mathsf{in}^{\ell^t}\mu \in \mathrm{CAPA}_{me_\star}(P_\star)$. Assume that $(in\,\mu, \ell^a) \in \mathcal{F}(me_\star, P_\star, I)$, $(\ell^{a''}, \ell^a) \in I$ and $(\ell^{a''}, \ell^{a'}) \in I$. From Proposition 6.15 we have that $(\ell^{a'}, \ell^a) \in I$, so $(I, \mathcal{F}(me_\star, P_\star, I)) \models_c cc$. The cases for $cc = \{l^a\} \subseteq \mathtt{F}(out\,\mu) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})$ and $cc = \{l^a\} \subseteq \mathtt{F}(open\,\mu) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^{a'}\} \subseteq \mathtt{I}(l^{a''}) \texttt{ => } \{l^a\} \subseteq \mathtt{I}(l^{a'})$ are analogous. ∎

Proposition 6.19 *We have for all I and H*

$$\begin{aligned}&\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge\ (I,H) \models_{me_\star} P_\star \Rightarrow \\ &\quad (I,\mathcal{F}(me_\star,P_\star,I)) \models_c (\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star,H)[\![P_\star]\!])\end{aligned}$$

Proof. The first conjunction follows by an argument analogous to the proof of Proposition 6.8. The second conjunction follows from Lemma 6.18 ■

Now, in order to find the least analysis to $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \wedge (I,H) \models_{me_\star} P_\star$ we find the least solution (I,F) to the set of constraints generated by $\mathcal{C}_\beta$ and $\mathcal{C}'_{\models}$, $(\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star,\mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!])$, with respect to the ordering:

$$(I_1,F_1) \sqsubseteq_{\mathbf{IF}} (I_2,F_2) \text{ iff } I_1 \subseteq I_2 \ \wedge\ F_1 \subseteq F_2$$

The least solution can be obtained by extending the algorithm of Table 6.5. Having found I as part of the least solution, we will now show that $(I,\mathcal{H}(me_\star)[\![P_\star]\!])$ is the least analysis by the following proposition.

Proposition 6.20 *We have for all $I \subseteq I^\top_\star$ and $H \subseteq \mathcal{H}(me_\star)[\![P_\star]\!]$: if (I,F) is the least solution to*

$$(I,F) \models_c (\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star,H)[\![P_\star]\!])$$

then I is the least solution to

$$\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge\ (I,H) \models_{me_\star} P_\star$$

Proof. Let $I \subseteq I^\top_\star$ and $H \subseteq \mathcal{H}(me_\star)[\![P_\star]\!]$ be given. Assume that (I,F) is the least solution to $(I,F) \models_c (\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}_{\models}(me_\star,H)[\![P_\star]\!])$. By Proposition 6.17 we have that I is a solution to $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge\ (I,H) \models_{me_\star} P_\star$. Assume per absurdum that I is not the least solution. Then there exists I' such that $I' \subset I$ and $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I',H) \ \wedge\ (I',H) \models_{me_\star} P_\star$. By 6.19 we have that $(I',\mathcal{F}(me_\star,P_\star,I')) \models_c (\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star,H)[\![P_\star]\!])$. This is a contradiction since $(I,F) \not\sqsubseteq_{\mathbf{IF}} (I',\mathcal{F}(me_\star,P_\star,I'))$ hence I is the least solution to $\beta^{\ell^a_\star}_{me_\star}(P_\star) \sqsubseteq (I,H) \ \wedge\ (I,H) \models_{me_\star} P_\star$. ■

In order to determine the time needed to find the least analysis we examine the number of generated constraints. From $\mathcal{C}_\beta(\ell^a_\star)[\![P_\star]\!] \cup \mathcal{C}'_{\models}(me_\star,\mathcal{H}(me_\star)[\![P_\star]\!])[\![P_\star]\!]$, $O(s^3)$ constraints are generated if the size of $P_\star$ is $O(s)$. To see this note that we generate $O(s)$ constraints in the form:

$$\{l^{at}\} \subseteq \mathtt{I}(l^a)$$

and $O(s^2)$ constraints in the form:

$$\{l^t\} \subseteq \mathtt{I}(l^a) \texttt{ => } \{l^a\} \subseteq \mathtt{F}(\mathtt{c})$$

where $\mathtt{c} \in \{in\,\mu, out\,\mu, open\,\mu\}$. Finally for each μ in

$$\left\{\mu \,\middle|\, \exists \ell^t \in \textsf{\textbf{Lab}}^t : \left\{\textsf{in}^{\ell^t}\mu, \textsf{out}^{\ell^t}\mu, \textsf{open}^{\ell^t}\mu\right\} \cap \mathrm{CAPA}_{me_\star}(P_\star) \neq \emptyset\right\}$$

we generate at most $3 \cdot O(s^2) \cdot |\{\ell^a \,|\, (\ell^a, \mu) \in \mathcal{H}(me_\star)[\![P_\star]\!]\}|$ constraints of the form:

$$\{l_1^a\} \subseteq \mathtt{F(c)} \texttt{ => } \{l_2^{at}\} \subseteq \mathtt{I}(l_2^a) \texttt{ => } \{l_3^{at}\} \subseteq \mathtt{I}(l_3^a) \texttt{ => } \{l^{at}\} \subseteq \mathtt{I}(l^a)$$

and we can put a bound on the number of these constraints by:

$$\sum_{\mu \in \textsf{\textbf{Mar}}} 3 \cdot O(s^2)\, |\{\ell^a \,|\, (\ell^a, \mu) \in \mathcal{H}(me_\star)[\![P_\star]\!]\,\}| = O(s^3)$$

Following the development of Section 6.4.1, the size of the set of occurring constants u is $|\textsf{\textbf{Lab}}_\star| = O(s)$ and as shown above the number of constraints is $t = O(s^3)$. We conclude that the constraints can be solved in time $O(s^4)$ using an extended version of the algorithm.

Chapter 7

Prototype Implementation

In the preceding chapters we have seen a number of concepts that are readily computed. In Chapter 5 we introduced a number of static checks that implied dynamic counterparts. Most of these checks involves finding an analysis that fulfils certain requirements. In most cases the least analysis is crucial to the check in the sense that the static check holds if and only if the least analysis fulfils the requirements. In Chapter 6 an algorithm to find the least solution was presented. Computing the least solution and examining whether static checks hold or not soon becomes a tedious and error prone activity if it is carried out by hand. Therefore a prototype system has been constructed to support the understanding of the introduced concepts. The user can write a process as in [CG98], i.e. without labels and markers and have the least solution computed as well as having checked whether the process is a firewall or not (in the style of [NNHJ99]).

The aim of this chapter is two-fold. Firstly it will present the prototype at an abstraction level where it may aid people who are interested in implementing a similar system. Secondly it will serve as a guide to the web interface that has been implemented in order to give easy access to the system. If the reader only wants to use the web interface to get a feel of the analysis and the related concepts he should at least read Section 7.1.1 where the input syntax is explained before moving on to Section 7.2.

7.1 The underlying system

The basis of the prototype is an implementation in Moscow ML[1] of a system that consists of

- a parser for the mobile ambient calculus where terms are described without use of labels and markers.

[1] See `http://www.dina.kvl.dk/~sestoft/mosml.html` for more information.

- an annotator that automatically annotates a term with labels and markers.
- a generator that generates constraints in the manner of the $\mathcal{C}_\beta$ and $\mathcal{C}_\models$ functions of Tables 6.2 and 6.3.
- a constraint solver that implements the algorithm of Table 6.5.
- a function that determines whether a given process is a firewall in the style of [NNHJ99].

In the following we will describe the individual constituents of the system.

7.1.1 The parser

The task of the parser is to process a basic text string and from this produce a data value, provided the text string represents a valid process. The generated data value conveys information on how the process could be produced by the grammar of Chapter 3 (not taking into account labels and markers).

The external representation of a process

We will now show how to write processes. The syntax can be described by a grammar similar to the one describing processes in Chapter 3:

$$\begin{array}{lcl@{\qquad}lcl}
\mathbf{P},\mathbf{Q} & ::= & \texttt{(/n)}\mathbf{P} & \mathbf{M} & ::= & \texttt{in n} \\
 & | & \texttt{0} & & | & \texttt{out n} \\
 & | & \mathbf{P}\ \texttt{|}\ \mathbf{Q} & & | & \texttt{open n} \\
 & | & \texttt{!}\mathbf{P} & & & \\
 & | & \texttt{n[}\mathbf{P}\texttt{]} & & & \\
 & | & \mathbf{M}\texttt{.}\mathbf{P} & & &
\end{array}$$

where `n` represents an identifier (has to begin with a letter). The parser allows the abbreviation $\mathbf{M} \triangleq \mathbf{M}\texttt{.0}$ and parses according to the notational conveniences mentioned in Chapter 3:

$$\begin{array}{lll}
\texttt{(/n)}\mathbf{P}\ \texttt{|}\ \mathbf{Q} & \text{is parsed as} & \texttt{((/n)}\mathbf{P}\texttt{)}\ \texttt{|}\ \mathbf{Q} \\
\texttt{!}\mathbf{P}\ \texttt{|}\ \mathbf{Q} & \text{is parsed as} & \texttt{(!}\mathbf{P}\texttt{)}\ \texttt{|}\ \mathbf{Q} \\
\mathbf{M}\texttt{.}\mathbf{P}\ \texttt{|}\ \mathbf{Q} & \text{is parsed as} & \texttt{(}\mathbf{M}\texttt{.}\mathbf{P}\texttt{)}\ \texttt{|}\ \mathbf{Q}
\end{array}$$

Example 7.1 The simple processes

$$\mathbf{0},\ (\nu n)\mathbf{0},\ n[\mathbf{0}] \text{ and } in\, m$$

should be written to the parser as

`0`, `(/n)0`, `n[0]` and `in m`

respectively ◇

Example 7.2 The process of Example 3.1, where we leave out the labels and markers, is the process

$$(\nu m)(n[in\, m] \mid m[open\, n])$$

The corresponding piece of syntax is:

```
(/m)(n[in m] | m[open n])
```

This will be used in the following examples to illustrate the workings of the different parts of the prototype. ◇

The internal representation of a process

Now that the textual representation of a process in the system has been shown it is time to describe the way a process is represented internally. The parser produces a value of the datatype `Proc`:

```
type ThrLabel = string                            (* Thread labels  *)
type AmbLabel = string                            (* Ambient labels *)
type Marker = string                              (* Markers        *)
type Name = string                                (* Ambient names  *)

datatype Proc = Nu of Marker*Name*Proc            (* Restriction    *)
              | Zero                              (* Inactivity     *)
              | Par of Proc*Proc                  (* Composition    *)
              | Bang of Proc                      (* Replication    *)
              | Amb of Name*AmbLabel*Proc         (* Ambient        *)
              | Incap of Name*ThrLabel*Proc       (* Enter n        *)
              | Outcap of Name*ThrLabel*Proc      (* Exit n         *)
              | Opencap of Name*ThrLabel*Proc     (* Open n         *)
```

This is the representation of a process used by the whole system so labels and markers are also included. The parser will just insert arbitrary markers and labels leaving it to the annotator to give meaningful markers and labels to the process; the implemented parser inserts the empty string for all markers and labels.

Example 7.3 Assume that the system is running. In the computer listings lines starting with "`-`" are user input whereas lines starting with "`>`" are output from the system. We supply the parser `parser` with a correct textual representation of a process, namely the string from Example 7.2:

```
- val exampleparsed = parser "(/m)(n[in m] | m[open n])";
> val exampleparsed =
    Nu("", "m",
       Par(Amb("n", "", Incap("m", "", Zero)),
           Amb("m", "", Opencap("n", "", Zero))))
    : Proc
```

We see that the parser accepts the string and produces the appropriate `Proc`-value. ◇

Example 7.4 We now supply the parser with a text string that does not represent a process:

```
- parser "n[0[";
! Uncaught exception:
! syntax_error "Could not find matching ] at n [0 ["
```

The system reports an error since the ambient is not ended by “`]`”. ◇

When we have used the parser on a textual representation of a process we have gone from a text string to an internal representation of a process that can be used by the later stages of the system. The next stage will be the annotation of the value obtained by the parser.

7.1.2 The annotator

The task of the annotator is to insert labels and markers in a `Proc` value obtained by the parser. The input to the annotator is a `Proc` value and the output is a new `Proc` value with inserted labels and markers according to a certain convention. The markers and labels used by the implemented annotator is

- markers: `"m1"`, `"m2"`, ...
- ambient labels: `"a1"`, `"a2"`, ...
- thread labels: `"t1"`, `"t2"`, ...

The workings of the annotator are best illustrated by an example:

Example 7.5 If we supply the annotator with the `Proc` value obtained from the parser in Example 7.3 we obtain:

```
- val exampleannotated = annotate exampleparsed;
    Nu("m1", "m",
       Par(Amb("n", "a1", Incap("m", "t1", Zero)),
           Amb("m", "a2", Opencap("n", "t2", Zero))))
```

The restriction is annotated with `m1`, the ambients n and m are annotated with labels `a1` and `a2` and the capabilities $in\,m$ and $out\,n$ are annotated with `t1` and `t2`.

Example 3.1	Example 7.5
μ	`m1`
ℓ^a_n	`a1`
ℓ^a_m	`a2`
ℓ^t_1	`t1`
ℓ^t_2	`t2`

The above table lists the annotations from Example 3.1 and the automatically generated annotations by the system.

$\diamond$

In order to assign markers to the free names of a process, the prototype implements the function `makeme` which generates a marker environment for a given `Proc`-value. Each free name in the process is associated with a marker which is simply the name prefixed by `m`. It follows that if the `Proc`-value has been annotated by the `annotate`-function above then `makeme` binds all free names to fresh markers.

Example 7.6 Here we will create a marker environment for the `Proc`-value in Example 7.3:

```
- val me = makeme exampleparsed;
> val me = <env> : (string, string) Env.env
- val melist = Env.tolist me;
> val melist = [("n", "mn")] : (string * string) list
```

The only free name of the process, `n`, is associated with the marker `mn`. $\diamond$

We now have a representation of a process described by the grammar of Chapter 3 together with a marker environment. In the next section we show how the development of finding a least analysis in Chapter 6 has been incorporated into the system.

7.1.3 The generator of constraints

In Chapter 6 it was described how to find the least solution to $\beta^{\ell^a}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$. This was done in several steps. The least H was proved to be $\mathcal{H}(me)[\![P]\!]$ and the least I was proved to be the least solution to the set of constraints $\mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \mathcal{C}_\models(me, H)[\![P]\!]$. The generator of constraints in the prototype is responsible for computing $\mathcal{H}(me)[\![P]\!]$ and $\mathcal{C}_\beta(\ell^a)[\![P]\!] \cup \mathcal{C}_\models(me, H)[\![P]\!]$.

The constraints in the system are of the datatype `Constr`:

	Ambient labels			Thread labels		
Integer value	0	$\cdots$	$k-1$	k	$\cdots$	$k+m-1$
Constant	l_1^a	$\cdots$	l_k^a	l_1^t	$\cdots$	l_m^t
Variable	$\mathtt{I}(l_1^a)$	$\cdots$	$\mathtt{I}(l_k^a)$			

Table 7.1: Association between integers, constants and variables

```
datatype CVal    = Var of int
                 | Cst of int list

datatype Constr  = Subset of CVal * CVal
                 | Cnd of Constr list
```

The `Constr` data type is actually too general when compared to the grammar presented in Section 6.3. It is important to point out that only constraints abiding to the grammar of Section 6.3 will be generated by the system and only sets of this kind of constraints will be solved by the system. The reason to choose a more general data type is just one of convenience.

The variables generated by $\mathcal{C}_\beta$ and $\mathcal{C}_\models$ are represented by integers from 0 to $k-1$ where k is the number of variables and the generated constants are represented by integers from 0 to $l-1$ where l is the number of constants. If $\mathsf{Lab}_\star^a = \ell_1^a, \ldots, \ell_k^a$ and $\mathsf{Lab}_\star^t = \ell_1^t, \ldots, \ell_m^t$ then the association is as shown in Table 7.1.

The function `make_constraints` takes a process P with initial marker environment me and enclosing ambient label ℓ^a, computes $\mathcal{H}(me)[\![P]\!]$ and produces constraints corresponding to $\mathcal{C}_\beta(\ell^a)[\![P]\!]$ and $\mathcal{C}_\models(me, \mathcal{H}(me)[\![P]\!])[\![P]\!]$.

Example 7.7 We will generate constraints from the `Proc`-value of Example 7.3 and the marker environment of Example 7.6. The constraints are generated by the following expression where `a` associates integer values to constants and variables; `maxvar` is the number of distinct variables minus one; `beta` corresponds to $\mathcal{C}_\beta(\ell^a)[\![P]\!]$ (where ℓ^a is represented by `"top"`) and `models` corresponds to $\mathcal{C}_\models(me, \mathcal{H}(me)[\![P]\!])[\![P]\!]$:

```
- val (a,maxvar,beta,models) =
            make_constraints "top" me exampleannotated;
> val a = <assoc> : string assoc
  val maxvar = 3 : int
  val beta =
    [Subset(Cst [0], Var 2), Subset(Cst [3], Var 0),
     Subset(Cst [1], Var 2), Subset(Cst [4], Var 1)]
    : Constr list
  val models =
```

```
[Cnd [Subset(Cst [3], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [1], Var 0), Subset(Cst [0], Var 1)],
 Cnd [Subset(Cst [3], Var 0), Subset(Cst [0], Var 1),
      Subset(Cst [1], Var 1), Subset(Cst [0], Var 1)],
 Cnd [Subset(Cst [3], Var 0), Subset(Cst [0], Var 2),
      Subset(Cst [1], Var 2), Subset(Cst [0], Var 1)],
 Cnd [Subset(Cst [3], Var 1), Subset(Cst [1], Var 0),
      Subset(Cst [1], Var 0), Subset(Cst [1], Var 1)],
 Cnd [Subset(Cst [3], Var 1), Subset(Cst [1], Var 1),
      Subset(Cst [1], Var 1), Subset(Cst [1], Var 1)],
 Cnd [Subset(Cst [3], Var 1), Subset(Cst [1], Var 2),
      Subset(Cst [1], Var 2), Subset(Cst [1], Var 1)],
 Cnd [Subset(Cst [3], Var 2), Subset(Cst [2], Var 0),
      Subset(Cst [1], Var 0), Subset(Cst [2], Var 1)],
 Cnd [Subset(Cst [3], Var 2), Subset(Cst [2], Var 1),
      Subset(Cst [1], Var 1), Subset(Cst [2], Var 1)],
 Cnd [Subset(Cst [3], Var 2), Subset(Cst [2], Var 2),
      Subset(Cst [1], Var 2), Subset(Cst [2], Var 1)],
 Cnd [Subset(Cst [4], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [0], Var 0), Subset(Cst [0], Var 0)],
 Cnd [Subset(Cst [4], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [1], Var 0), Subset(Cst [1], Var 0)],
 Cnd [Subset(Cst [4], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [2], Var 0), Subset(Cst [2], Var 0)],
 Cnd [Subset(Cst [4], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [3], Var 0), Subset(Cst [3], Var 0)],
 Cnd [Subset(Cst [4], Var 0), Subset(Cst [0], Var 0),
      Subset(Cst [4], Var 0), Subset(Cst [4], Var 0)],
 Cnd [Subset(Cst [4], Var 1), Subset(Cst [0], Var 1),
      Subset(Cst [0], Var 0), Subset(Cst [0], Var 1)],
 Cnd [Subset(Cst [4], Var 1), Subset(Cst [0], Var 1),
      Subset(Cst [1], Var 0), Subset(Cst [1], Var 1)],
 Cnd [Subset(Cst [4], Var 1), Subset(Cst [0], Var 1),
      Subset(Cst [2], Var 0), Subset(Cst [2], Var 1)],
 Cnd [Subset(Cst [4], Var 1), Subset(Cst [0], Var 1),
      Subset(Cst [3], Var 0), Subset(Cst [3], Var 1)],
 Cnd [Subset(Cst [4], Var 1), Subset(Cst [0], Var 1),
      Subset(Cst [4], Var 0), Subset(Cst [4], Var 1)],
 Cnd [Subset(Cst [4], Var 2), Subset(Cst [0], Var 2),
      Subset(Cst [0], Var 0), Subset(Cst [0], Var 2)],
 Cnd [Subset(Cst [4], Var 2), Subset(Cst [0], Var 2),
      Subset(Cst [1], Var 0), Subset(Cst [1], Var 2)],
 Cnd [Subset(Cst [4], Var 2), Subset(Cst [0], Var 2),
```

```
          Subset(Cst [2], Var 0), Subset(Cst [2], Var 2)],
     Cnd [Subset(Cst [4], Var 2), Subset(Cst [0], Var 2),
          Subset(Cst [3], Var 0), Subset(Cst [3], Var 2)],
     Cnd [Subset(Cst [4], Var 2), Subset(Cst [0], Var 2),
          Subset(Cst [4], Var 0), Subset(Cst [4], Var 2)]]
    : Constr list
```

The association between integers and labels presented in Table 7.1 is given by `a` above. Here the association is presented in a readable form:

```
- val association = Assoc.aslist a;
> val association =
    [(0, "a1"), (1, "a2"), (2, "top"), (3, "t1"), (4, "t2")]
    : (int * string) list
```

Written in the style of Table 7.1 the above represents:

	Ambient labels			Thread labels	
Integer value	0	1	2	3	4
Constant	`a1`	`a2`	`top`	`t1`	`t2`
Variable	`I(a1)`	`I(a2)`	`I(top)`		

The constraints of Example 6.7 were produced by pretty-printing functions generating LaTeX from the above constraints. The labels in the LaTeX-code were modified manually according to the table of Example 7.5 ◇

Having generated the constraints we are just one step away from obtaining a least solution: actually solving the constraints.

7.1.4 The solver

The solver implements the algorithm of Table 6.5. It makes use of Moscow ML's imperative features to represent W, D and E. The system offers the possibility of outputting a trace of the run of the solver where W,D and E are reported for each time the algorithm goes through the while loop in Step 3. Table 6.7 was constructed using this feature.

Example 7.8 In order to find the least solution to the process of Example 3.1, we will solve the constraints that were generated in Example 7.7.

```
- val (examplesolved,_) = solution a maxvar (beta@models) false;
> val examplesolved = <setfunc> : (string, string) Setfunc.setfunc
- val solved = Setfunc.listfunc examplesolved;
> val solved =
    [("a1", ["t1"]), ("a2", ["a1", "a2", "t1", "t2"]),
     ("top", ["a1", "a2"])] : (string * string list) list
```

The least analysis presented in Example 6.11 was derived from this solution. ◇

7.1.5 Firewall

In [NNHJ99] it was studied whether a given process was a firewall, i.e. whether it allows unauthorised processes (not knowing certain keys) to enter a certain ambient (the firewall) or not. The check of the firewall property has been implemented by the function `isfirewall`.

Example 7.9 Here we will show that the firewall process introduced in Example 3.2 is a firewall in the sense of [NNHJ99]. The firewall should be presented to the parser as (where the generic process P has been instantiated):

```
"w[k[out w.in k'.in w]|open k'.open k''.
                (!p[in p|out p|open p|p[0]])]"
```

In the following computer listing it is shown how the function `isfirewall` is applied to the process.

```
- val firewall = "w[k[out w.in k'.in w]|open k'.open k''.
                                (!p[in p|out p|open p|p[0]])]";
> val firewall =
    "w[k[out w.in k'.in w]|open k'.open k''.
                                (!p[in p|out p|open p|p[0]])]"
    : string
- isfirewall "top" (annotate(parser(firewall)));
> val it = true : bool
```

The system responds `true` signifying that the process is indeed a firewall. $\diamond$

7.2 The webprototype

In order to allow easy access to the prototype, we have implemented a web-based interface, which can be accessed at `http://www.daimi.au.dk/~grydholt/macfa.html`. On this page there is some information on the system together with a description of the syntax. In this section we will go into greater detail on the workings of this part of the prototype. By following the link "Starting the system" the user gets a web page as shown in Table 7.2. The page is divided into two frames. The upper frame is the entry part where the process of interest is selected or typed in. When the process has been chosen by the user, the system parses it and outputs it in the lower frame along with a list of actions that can be performed on it.

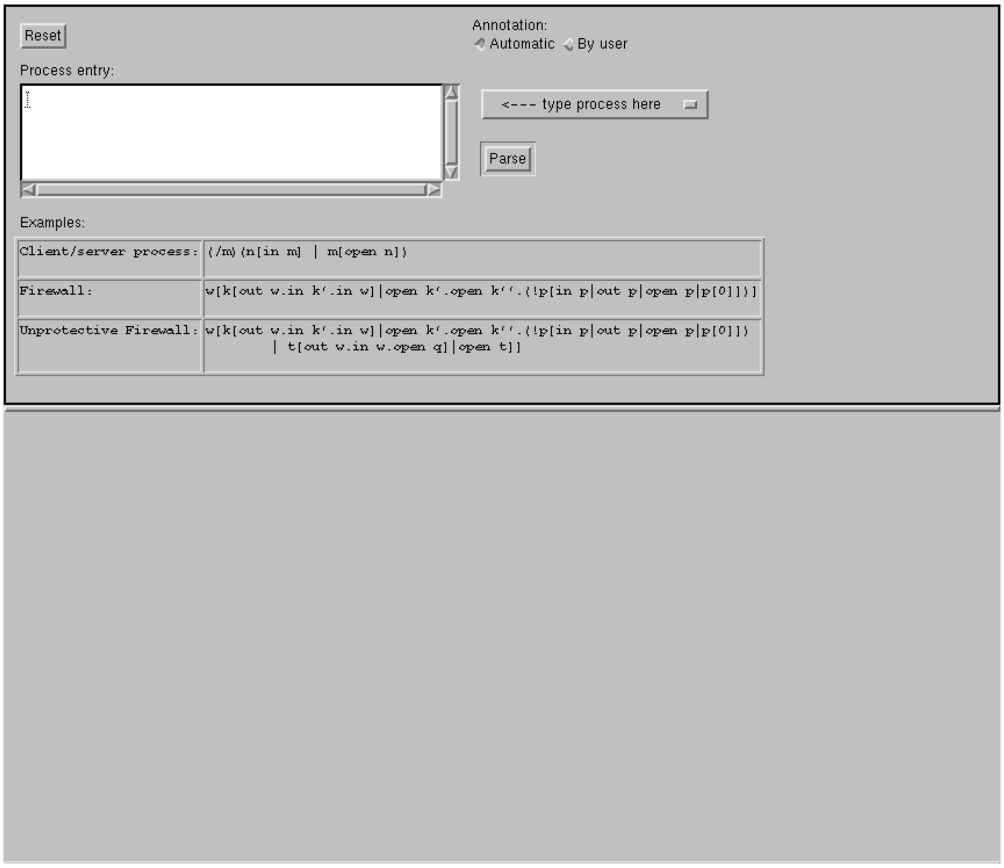
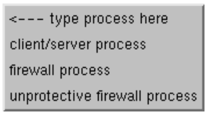


Table 7.2: The start page

7.2.1 Choosing a process

The user can choose to type in his own process or select a predefined example process. This is controlled by the menu to the right of the process entry field (see Table 7.2). If the user presses the menu he has the following choices:



Selecting the first item allows the user to type his own process in the entry field. By selecting another item he chooses one of the predefined example processes at the bottom of the entry frame. The "client/server process" is the running example from this thesis. The other processes were used in [NNHJ99] as example of respectively a protective and an unprotective firewall.

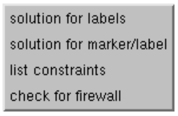
The user can choose to let the system annotate the process by setting the annotation to "Automatic" or choose to have full control of the annotation by setting the annotation to "By user".

When a process has been typed in or chosen, the user presses the "Parse" button to have the process parsed and annotated as described in Section 7.1.1 and Section 7.1.2. The output of the system is presented in the lower frame.

To illustrate the use of the webprototype, we will in the following assume that the user has selected the "client/server process". If the automatic annotation is used, the output is as shown in Table 7.3, whereas if user-defined annotation was chosen, the output would be presented as shown in Table 7.4. The output is initially the same, since both outputs will have been automatically annotated, but with the user-defined annotation there is the possibility of changing markers and labels in the appropriate entry fields.

7.2.2 Choosing an action

The user should inspect the values of P, ℓ^a and me presented in the lower frame to ensure himself that they are correct. If this is the case the user proceeds by selecting an action from the "Action" menu:



The first two items presents the least solution to $\beta^{\ell^a}_{me}(P) \sqsubseteq (I, H) \wedge (I, H) \models_{me} P$ in different ways; "solution for labels" presents the full solution, whereas "solution for marker/label" presents the solution with respect to a single marker or label. The action "list constraints" shows the constraints that when solved gives the

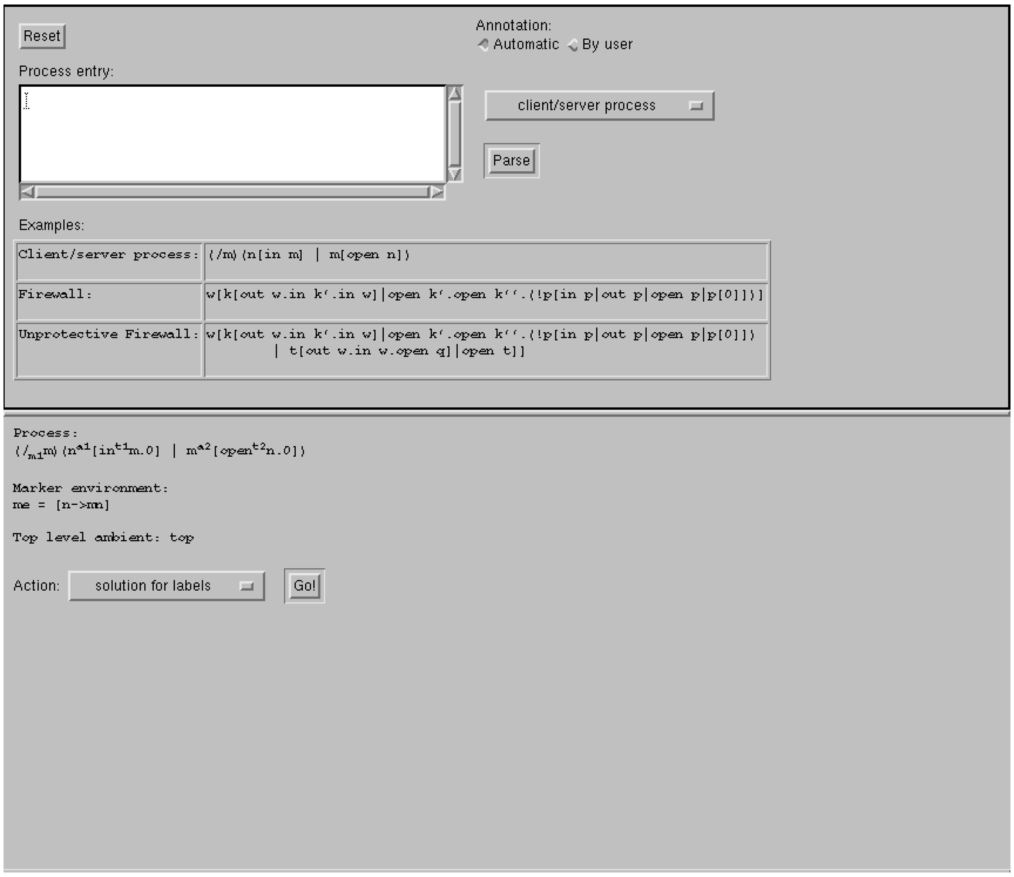


Table 7.3: Automatic annotation of process

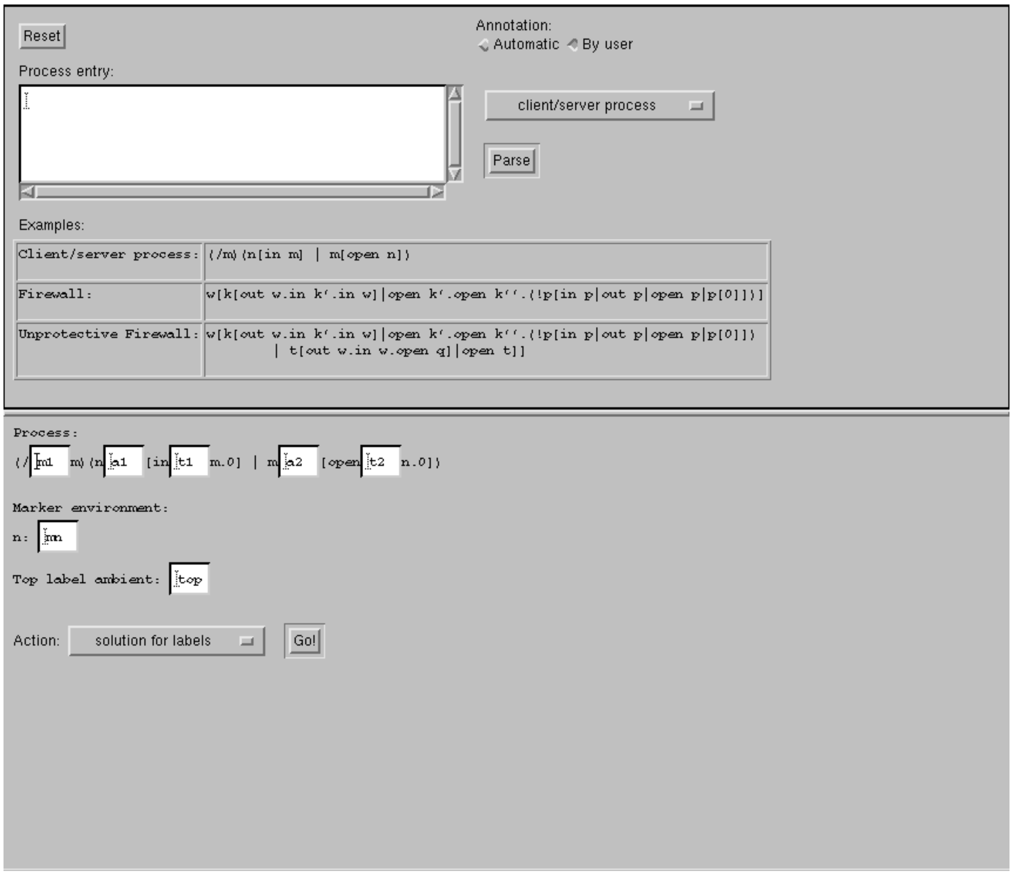
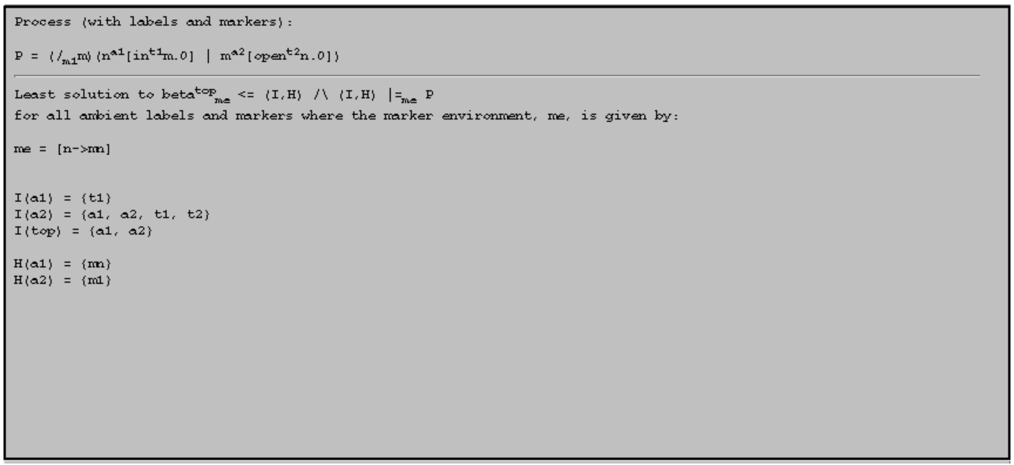


Table 7.4: User-defined annotation of process



Process (with labels and markers):

P = $(/_{m1}m)(n^{a1}[in^{t1}m.0]$ | $m^{a2}[open^{t2}n.0])$

Least solution to $beta^{top}{}_{me}$ <= (I,H) /\ (I,H) $|=_{me}$ P
for all ambient labels and markers where the marker environment, me, is given by:

me = [n->mn]

I(a1) = {t1}
I(a2) = {a1, a2, t1, t2}
I(top) = {a1, a2}

H(a1) = {mn}
H(a2) = {m1}

Table 7.5: Output for the "solution for labels" action

I-component of the solution presented by the first two items. Finally "check for firewall" gives the possibility of checking whether the process is a firewall or not. The selected action is performed by clicking the "Go!"-button next to the "Action" menu.

7.3 Result of the actions

Now we will present the results that can be obtained by selecting the different actions. We assume that the user has selected the "client/server-process" and automatic annotation.

7.3.1 The "solution for labels" action

By choosing the "solution for labels" action the system presents the least solution to $\beta^{\ell^a}_{me}(P) \sqsubseteq (I,H) \ \wedge \ (I,H) \models_{me} P$ in the output frame as shown in Table 7.5. Here $I(\ell^a)$ stands for $\{\ell^{at} \,|\, (\ell^a, \ell^{at}) \in I\}$ and likewise $H(\ell^a)$ stands for $\{\mu \,|\, (\ell^a, \mu) \in H\}$. This information was also presented in a slightly different way in Examples 6.11 and 6.3.

7.3.2 The "marker/label" action

Choosing "marker/label" the user has the possibility to only see the part of a solution pertaining to a single marker or label. After a click on the "Do it!"-button the output frame looks like shown in Table 7.6. Here the markers and ambient labels act as links to parts of the solution. Clicking on a marker μ gives as result $H^{-1}(\mu)$ which is short for $\{\ell^a \,|\, (\ell^a, \mu) \in H\}$, i.e. the ambient labels bound at the selected marker. Table 7.7 shows the output frame after the user has

Process (with labels and markers):

$(/_{\underline{m1}} m)(n^{\underline{a1}}[in^{t1} m.0] \mid m^{\underline{a2}}[open^{t2} n.0])$

Table 7.6: Output for the "solution for marker/label" action

Process (with labels and markers):

$P = (/_{\underline{m1}} m)(n^{\underline{a1}}[in^{t1} m.0] \mid m^{\underline{a2}}[open^{t2} n.0])$

Least solution to $beta^{top}_{me}$ <= (I,H) /\ (I,H) $|=_{me}$ P for marker
m1 where the marker environment, me, is given by:

me = [n->mn]

$H^{-1}(m1) = \{a2\}$

Table 7.7: The solution relative to particular marker

Process (with labels and markers):

P = $(/_{\underline{m1}}m)(n^{\underline{a1}}[in^{t1}m.0] \mid m^{\underline{a2}}[open^{t2}n.0])$

Least solution to $beta^{top}_{me}$ <= (I,H) /\ (I,H) $|=_{me}$ P
for ambient label a2 where the marker environment, me, is given by:

me = [n->mn]

I(a2) = {a1, a2, t1, t2}

Table 7.8: The solution relative to a particular ambient

clicked on the marker "`m1`". the output frame will change as shown in Table 7.7. Here $H^{-1}(\mu)$ stands for $\{\ell^a \mid (\ell^a, \mu) \in H\}$, i.e. the ambient labels bound at the selected marker.

By clicking on an ambient label ℓ^a the user obtains information on which ambients and capabilities that may end up inside the ambient, i.e. $\{\ell^{at} \mid (\ell^a, \ell^{at}) \in I\}$ represented by $I(\ell^a)$. Table 7.8 shows the output frame after the user has clicked on the ambient label "`a2`'.

7.3.3 The "list constraints" action

If "list constraints" is chosen the output frame displays the sets of constraints $\mathcal{C}_\beta(\ell^a)[\![P]\!]$ and $\mathcal{C}_{\models}(me, \mathcal{H}(me)[\![P]\!])[\![P]\!]$ as shown in Table 7.9. These constraints have also been presented in different forms in Examples 6.7 and 7.7.

7.3.4 The "check for firewall" action

The "check for firewall" action checks whether the process is a firewall in the sense of [NNHJ99]. Not surprisingly the prototype reports that the example process "Firewall" is a firewall whereas the example process "Unprotective Firewall" is not.

Process (with labels and markers):

$P = (/_{m1}m)(n^{a1}[in^{t1}m.0] \mid m^{a2}[open^{t2}n.0])$

Constraints C_{beta}(top)[P]:

```
{a1} <= I(top)
{t1} <= I(a1)
{a2} <= I(top)
{t2} <= I(a2)
```

Constraints $C_{|=}$(me,H)[P] where the marker environment, me, is given by:

```
me = [n->mn]
```

And H is given by:

```
H(a1) = {mn}
H(a2) = {m1}

{t1} <= I(a1) ==> {a1} <= I(a1) ==> {a2} <= I(a1) ==> {a1} <= I(a2)
{t1} <= I(a1) ==> {a1} <= I(a2) ==> {a2} <= I(a2) ==> {a1} <= I(a2)
{t1} <= I(a1) ==> {a1} <= I(top) ==> {a2} <= I(top) ==> {a1} <= I(a2)
{t1} <= I(a2) ==> {a2} <= I(a1) ==> {a2} <= I(a1) ==> {a2} <= I(a2)
{t1} <= I(a2) ==> {a2} <= I(a2) ==> {a2} <= I(a2) ==> {a2} <= I(a2)
{t1} <= I(a2) ==> {a2} <= I(top) ==> {a2} <= I(top) ==> {a2} <= I(a2)
{t1} <= I(top) ==> {top} <= I(a1) ==> {a2} <= I(a1) ==> {top} <= I(a2)
{t1} <= I(top) ==> {top} <= I(a2) ==> {a2} <= I(a2) ==> {top} <= I(a2)
{t1} <= I(top) ==> {top} <= I(top) ==> {a2} <= I(top) ==> {top} <= I(a2)
{t2} <= I(a1) ==> {a1} <= I(a1) ==> {a1} <= I(a1) ==> {a1} <= I(a1)
{t2} <= I(a1) ==> {a1} <= I(a1) ==> {a2} <= I(a1) ==> {a2} <= I(a1)
{t2} <= I(a1) ==> {a1} <= I(a1) ==> {top} <= I(a1) ==> {top} <= I(a1)
{t2} <= I(a1) ==> {a1} <= I(a1) ==> {t1} <= I(a1) ==> {t1} <= I(a1)
{t2} <= I(a1) ==> {a1} <= I(a1) ==> {t2} <= I(a1) ==> {t2} <= I(a1)
{t2} <= I(a2) ==> {a1} <= I(a2) ==> {a1} <= I(a1) ==> {a1} <= I(a2)
{t2} <= I(a2) ==> {a1} <= I(a2) ==> {a2} <= I(a1) ==> {a2} <= I(a2)
{t2} <= I(a2) ==> {a1} <= I(a2) ==> {top} <= I(a1) ==> {top} <= I(a2)
{t2} <= I(a2) ==> {a1} <= I(a2) ==> {t1} <= I(a1) ==> {t1} <= I(a2)
{t2} <= I(a2) ==> {a1} <= I(a2) ==> {t2} <= I(a1) ==> {t2} <= I(a2)
{t2} <= I(top) ==> {a1} <= I(top) ==> {a1} <= I(a1) ==> {a1} <= I(top)
{t2} <= I(top) ==> {a1} <= I(top) ==> {a2} <= I(a1) ==> {a2} <= I(top)
{t2} <= I(top) ==> {a1} <= I(top) ==> {top} <= I(a1) ==> {top} <= I(top)
{t2} <= I(top) ==> {a1} <= I(top) ==> {t1} <= I(a1) ==> {t1} <= I(top)
{t2} <= I(top) ==> {a1} <= I(top) ==> {t2} <= I(a1) ==> {t2} <= I(top)
```

Table 7.9: Output for the "list constraints" action

Chapter 8

Conclusion

In this thesis we have studied how static analysis in general, and flow logics in particular, can be applied to the ambient calculus. In Chapter 4 an analysis for the ambient calculus was specified in the framework of flow logics. The analysis statically approximates the movement of capabilities and ambients inside a given process.

In Chapter 5 the analysis was used to guarantee security properties for systems modelled in the ambient calculus. This was done by first identifying the dynamic property desired of the system at hand, and then defining a corresponding static notion sufficient to ensure the dynamic property. The static notion is defined by imposing conditions on an analysis of the system; this ensures the decidability of the static notion and allows for automated checking of the property in question.

In Chapter 6 we showed how to generate constraints equivalent to the specification of the analysis. A general observation allowed us to significantly reduce the number of constraints. Furthermore we presented a worklist algorithm that solves the constraints in low polynomial time. This led to the development of a web-based prototype implementing the worklist algorithm. The prototype, presented in Chapter 7, automatically generates and solves the constraints for a given process, and presents the result of the analysis. The result can furthermore be used to automatically check whether the process is a firewall.

Bibliography

[Aba97] Martin Abadi. Secrecy by Typing in Security Protocols. In *Proceedings Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638. Springer Verlag, 1997.

[AFFS98] Alexander Aiken, Manuel Fähndrich, Jeffrey S. Foster, and Zhendong Su. A Toolkit for Constructing Type- and Constrant-Based Program Analyses. In Xavier Leroy and Atsushi Ohori, editors, *Types in Compilation, Second International Workshop, TIC '98, Proceedings*, volume 1473 of *Lecture Notes in Computer Science*, pages 78–96, Kyoto, Japan, March 1998. Springer Verlag.

[AG98] Martin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. SRC Research Report 149, Digital Research, January 1998. URL:`http://www.digital.research.com/SRC/`.

[Aik94] Alexander Aiken. Set Constraints: Results, Applications and Future Directions. In Alan Borning, editor, *Principles and Practice of Constraints Programming, Second International Workshop, PPCP'94, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 326–335, Rosario, Orcas Island, WA, USA, May 1994. Springer Verlag.

[AKVW94] Alexander Aiken, Dexter Kozen, Moshe Vardi, and Edward Wimmers. The Complexity of Set Constraints. In Egon Börger, Yuri Gurevich, and Karl Meinke, editors, *7th Workshop, CSL'93*, volume 832 of *Lecture Notes in Computer Science*, pages 1–17. Springer Verlag, 1994.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers. Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[BDNN98] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control Flow Analysis for the π-calculus. In D. Sangiorgi and R. de Simone, editors, *Proceedingsa of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98, Nice, France, September 1998. Springer Verlag.

[BDNN99] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static Analysis of Processes for No Read-Up and No Write-Down. In Wolfgang Thomas, editor, *Foundations of Software Science and Computational Structures, Second International Conference (FoSSaCS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 120–134, Amsterdam, The Netherlands, March 1999. Springer Verlag. Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'99).

[CC92] Patrick Cousot and Radhia Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In Maurice Bruynooghe and Martin Wirsing, editors, *Proceedingsof PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, August 1992. Springer Verlag.

[CG97a] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. Draft, July 4 1997.

[CG97b] Luca Cardelli and Andrew D. Gordon. Mobile Ambients Annex. Draft, June 23 1997.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. In M. Nivat, editor, *Foundations of Software Science and Computation Structures First International Conference (FoSSaCS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, Lisbon, Portugal, March/April 1998. Springer Verlag. Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'98).

[CG99] Luca Cardelli and Andrew D. Gordon. Types for Mobile Ambients. In *Proceedings POPL'99*, pages 79–92. ACM Press, 1999.

[CGG99] Luca Cardelli, Giorgin Ghelli, and Andrew D. Gordon. Mobility Types for Mobile Ambients. In *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP'99)*, Lecture Notes In Computer Science, Prague, Czech Republic, July 1999. Springer Verlag. To appear.

[Che98] David M. Chess. Security Issues in Mobile Code Systems. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 1–14. Springer Verlag, 1998.

[Com95] Douglas E. Comer. *Internetworking with TCP/IP: Principles, Protocols and Architecture*, volume 1. Prentice Hall, 3rd edition, 1995.

[DD77] Dorothy Denning and Peter Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, 20(7):504–513, 1977.

[Den76] Dorothy Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–242, 1976.

[DP90] B.A. Davey and H.A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[EHM+99] Peter Harry Eidorff, Fritz Henglein, Christian Mossin, Henning Niss, Morten Heine Sørensen, and Mads Tofte. Annodomini: From type theory to year 2000 conversion tool. In *Proceedings of the 26th Symposium on the Principles of Programming Languages (POPL'99)*, 1999.

[GC99] Andrew D. Gordon and Luca Cardelli. Equational Properties of Mobile Ambients. In Wolfgang Thomas, editor, *Foundations of Software Science and Computational Structures, Second International Conference (FoSSaCS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 212–226, Amsterdam, The Netherlands, March 1999. Springer Verlag. Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'99).

[Hec77] Matthew S. Hecht. *Flow Analysis of Computer Programs*. Programming Language Series. North-Holland, 1977.

[Hei94] Nevin Heintze. Set-Based Analysis of ML Programs. In *Proceedings of the 1994 ACM Conference on LISP and Functional Programming*, pages 306–317, Orlando, Florida, June 1994. ACM.

[HJ94] Nevin Heintze and Joxan Jaffar. Set Constraints and Set-Based Analysis. In Alan Borning, editor, *Principles and Practice of Constraints Programming, Second International Workshop, PPCP '94, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, pages 281–298, Rosario, Orcas Island, WA, USA, May 1994. Springer Verlag.

[HJNN99] René R. Hansen, Jacob G. Jensen, Flemming Nielson, and Hanne Riis Nielson. Abstract Interpretation of Mobile Ambients. Submitted for publication, 1999.

[HR98] Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with Secrecy and Integrity. In *Conference Record of the Twenty-Fifth*

Annual ACM Symposium on Principles of Programming Languages, pages 365–377. ACM, 1998.

[KU77] John B. Kam and Jeffrey D. Ullman. Monotone Data Flow Analysis Frameworks. *Acta Informatica*, 7(3):305–317, 1977.

[Lan81] Carl E. Landwehr. Formal Models for Computer Security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[Mar98] Florian Martin. PAG – an efficient program analyzer generator. *International Journal of Software Tools for Technology Transfer*, 2(1):46–67, 1998.

[NN97] Flemming Nielson and Hanne Riis Nielson. A Collecting Semantics for Closure Analysis. In *Proceedings of the 24th Symposium on the Principles of Programming Languages (POPL'97)*, pages 332–345. ACM Press, 1997.

[NN98a] Flemming Nielson and Hanne Riis Nielson. Flow Logic and Operational Semantics. *Electronic Notes in Theoretical Computer Science*, 10, 1998. (URL: `http://www.elsevier.nl/locate/entcs/volume10.html`).

[NN98b] Flemming Nielson and Hanne Riis Nielson. The Flow Logic of Imperative Objects. In *Proceedings of Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 220–228. Springer Verlag, 1998.

[NNH] Flemming Nielson, Hanne Riis Nielson, and Chris L. Hankin. Principles of program analysis — flows and effects. preliminary version as of march 4, 1998; complete version to appear in 1999. (URL: `http://www.daimi.aau.dk/~hrn/PPA/ppa.html`).

[NNHJ99] Flemming Nielson, Hanne Riis Nielson, René R. Hansen, and Jacob G. Jensen. Validating Firewalls in Mobile Ambients. In *Proceedings of CONCUR'99*, Lecture Notes in Computer Science. Springer Verlag, July 1999. To appear.

[PS94] J. Palsberg and M.I. Schwartzbach. *Object-Oriented Type Systems.* Wiley, 1994.

[TBE+98] Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Højfeld Olesen, Peter Sestoft, and Peter Bertelsen. Programming with Regions in the ML Kit (For Version 3). Technical Report DIKU-TR-98/25, Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 KBH ,Denmark, December 1998.

[WM95] Reinhard Wilhelm and Dieter Maurer. *Compiler Design.* Addison-Wesley, 1995.