2006

A translator for automated code generation for service-based systems

Sanjib (Sean) Banerjee
West Virginia University

A TRANSLATOR FOR AUTOMATED CODE GENERATION FOR SERVICE-BASED SYSTEMS

Sanjib (Sean) Banerjee

Thesis submitted to the
College of Engineering & Mineral Resources at
West Virginia University
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Dr. Lan Guo
Dr. Supratik Mukhopadyay (Advisor/Committee Chair)
Dr. Frances Vanscoy

Lane Department of Computer Science

Morgantown, WV
2006

Keywords: Service Based Systems, Code Generation, Gentle, SOL, α-calculus

ABSTRACT

A Translator For Automated Code Generation For Service-Based Systems

Sanjib (Sean) Banerjee

Service based systems are playing a major role in industry today, everything from the verification of credit cards to booking airplane reservations are using some form of a service based approach. This unfortunately brings up a really big problem, how do we know that these services are actually doing what they are supposed to? Or, how do we know the service based system doesn't get somehow compromised when handling sensitive information.

A collaborative project involving Arizona State University and West Virginia University began in 2004 to first create a language, called α-calculus. We can prove that the code written in α-calculus adheres to the requirements. The next step was to create a translator that could convert code written in α-calculus to Secure Operations Language (SOL), which could then be used to develop service-based systems.

While research has been done in this area, the α-calculus to SOL translator provides a real world solution to creating provable/verifiable service based systems.

TABLE OF CONTENTS:

INTRODUCTION

This work reported here was supported by the DoD/ONR under the Multidisciplinary Research Program of the University Research Initiative under Contract No. N00014-04-1-0723. Not so long ago programs were written to perform, in today's terms, rather simplistic tasks. Programs usually did a few very specific tasks, interacting with a few hardware and software items. The dawn of the World Wide Web and interconnected systems heralded a new age of computer programming. Programs now not only had to interact with local hardware/software but also with hardware/software across a network. Sometimes this network was spread across states and countries, making the problems of security quite a nightmare.

Service based systems are playing a major role in industry today, everything from the verification of credit cards to booking airplane reservations are using some form of a service based approach. This unfortunately brings up a really big problem, how do we know that these services are actually doing what they are supposed to? Or, how do we know the service based system doesn't get somehow compromised when handling sensitive information. For instance, how do I know when I order that book from Amazon.com and hit the send button my credit card information is not being leaked out and being visible to everyone? Part of it is trust, we trust that the system Amazon.com or anyone else for that matter is using is a reliable one. Even then there are stories of how someone managed to gain access to private information. Surely we cannot base everything on trust, especially when talking about something even more critical like patient data at a hospital or a missile launch procedure. The question that arises is that is there a way we can ensure that our services do exactly what we want them to do?

Ensuring that programs do what they are supposed to do has been difficult given the nature of most programming languages. The languages we are generally used to, like C, C++, Java lack any formal semantics or the semantics are not useful. Therefore, it becomes very difficult to prove that your delivered product will actually do what you intended it to do. For instance, Microsoft certainly doesn't provide a certificate with their Windows CD that says that the product will do exactly what you want it to do without ever causing any unwanted reactions. Certainly the lack of such a certificate doesn't ward off potential clients, why because we assume that Microsoft has taken the time to extensively test their product and we won't encounter any problems that they haven't already found. This mentality doesn't translate very well when we start talking about systems like the stock exchange, patient monitoring systems, military systems, etc.

As an example let us look at a patient monitoring system, this system is being used to treat patients with thrombosis. It is a service based system and consists of a patient server, a central server, a nurse server, a dosage server and finally a doctor server. If a patient is in critical condition it triggers a flag and sends a request to the central server. The central server calls the doctor server stating that patient X is in critical condition. The doctor server in turn calls the dosage server requesting the correct dosage to help the patient. The value is returned from the dosage server back to the doctor server, the doctor server in turn returns the value to the central server. The central server then calls the nurse server which can then administer the dosage to help the patient.

Thrombosis is a medical condition that causes a blood clot to form within a blood vessel, thus obstructing the flow of blood within the circulatory system. Depending on the location of the thrombus, a patient could die if the condition is not properly treated. One of the medications used to treat thrombosis is Warfarin. Originally developed as rat poison, it has now found its place in the treatment of medical conditions in humans. Warfarin works by inhibiting the synthesis of blood clotting factors, in rodents this leads to death by excessive bleeding but in the treatment of thrombosis this actually benefits the patient. Drug monitoring is required because Warfarin has a very narrow therapeutic index, this means the levels that are effective and the levels that cause excessive bleeding are very close.

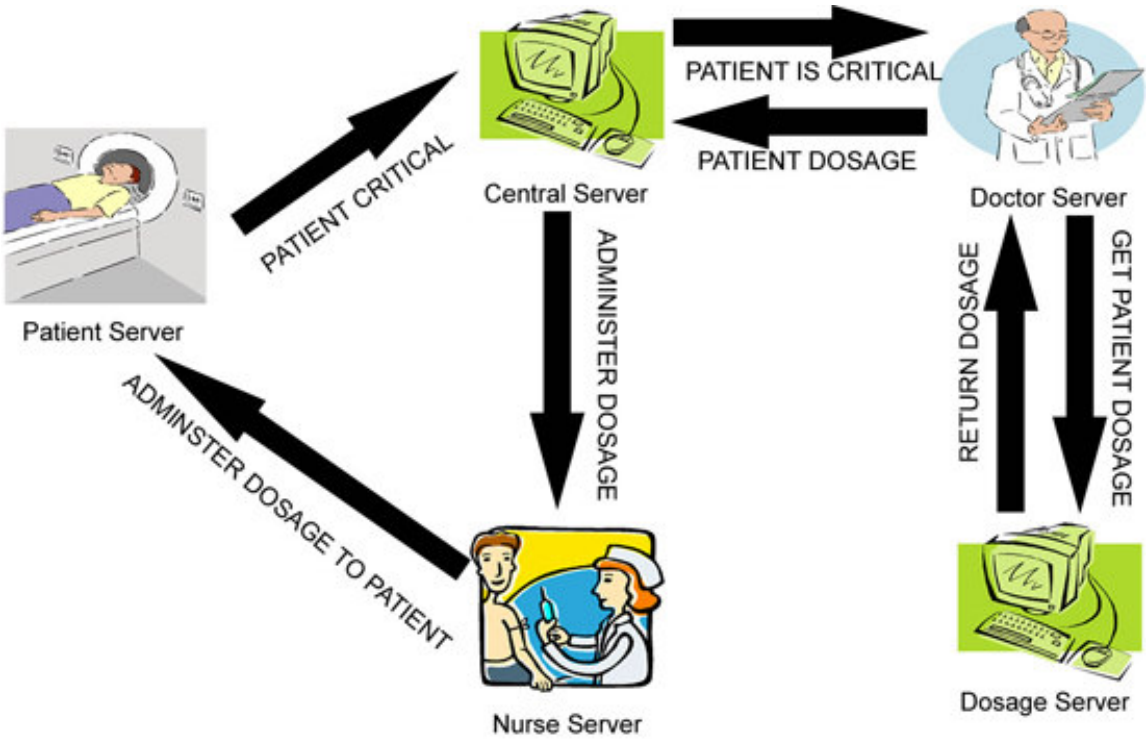

Figure 0: Patient Monitoring System

What if our patient monitoring system got compromised, instead of administering 5mg the patient gets 50mg. The consequences of this could certainly be the death of the patient; the domino effect could result in the hospital/doctor/etc getting sued for malpractice, etc. Certainly in a case like this failure is not an option, redundancy and fail safe methods are coded into the systems in case of a such a failure. But, that still doesn't guarantee that there won't be any incidents. As we move towards the next century the need for a more robust method of designing these mission critical service based systems is needed. The ideal solution would be one where the systems being developed can be proven to ensure their adhesion to the requirements.

Many papers have been written, and conferences have been held about creating some basic rules to achieve provability. After all it is the proverbial silver bullet that will end all problems. In the next section we take a look at one such paper written back in 1975, long before the World Wide Web era that talked about the need to create provable systems.

We need provability

Since the dawn of the programming era, provability has been one of the most sought after characteristics of a program. Provability refers to the validation of a program to meet certain specifications. This is not to be mistaken with reliability, which refers to a program's ability to conform to a certain expectations. In the 1970s Donald I. Good of The University of Texas at Austin wrote a paper titled Provable Programming where he delved into some of the ways a program could be made provable.

Good feels that there are five key principles to follow in order to write provable programs. First is creating specific specifications, this almost seems redundant but in order to prove a program we have to know exactly what it does. At the time specifications were rarely well written, strangely a problem that still exists today. Poorly written specifications cause code that may or may not meet the actual requirements of the client. Hence, it is vital to gather all the information first before proceeding into the coding phase. Next is the parallel development of program and proof, under this notion the proof of the program should be one of the design goals. The third aspect is proof factorization as Good calls it. This approach says that a program should not only be provable as a whole, but also provable as independent units. This is quite interesting, because it can almost be thought of as a pseudo proof by induction. You prove each piece of the code before proving the whole thing. Next is provability before efficiency, a program could do its task in 2 milliseconds but if we don't know for sure it actually does the task then it's rather useless. Lastly, the use of automated proof systems to help reduce the amount of manual proving a developer has to do. Given the complexity of most large scale programming projects, having to prove the whole thing by hand would take too long and be prone to mistakes. Having automation tools will certainly ease the problem, but care has to be taken to ensure the automation tools are correct in their methodology.

The ideas that Good puts forward would certainly work if the languages being used by the developers were provable. One could only do pseudo proofs that would show for n iterations of a program it did what it was supposed to do. This naturally brings up the question, what about the n+1 iteration would the program fail then? As programs became increasingly complex and with the advent of the World Wide Web, along with that service based systems and web services, the need to have provable programs became increasingly more urgent.

A collaborative project involving Arizona State University and West Virginia University began in 2004 to first create a language, called α-calculus. We can prove that the code written in α-calculus adheres to the requirements. The next step was to create a translator that could convert code written in α-calculus to Secure Operations Language (SOL), which could then be used to develop service-based systems. The idea was to first write code in α-calculus, prove it and then translate it to SOL and thus create a formal programming model that could be used to develop service based systems.

In the next section we will take a brief look at some of the languages used to create service systems, including the benefits and problems with each. Also, we will take a look at some of the existing formal approaches for developing service based systems.

Related work

The current industry standard for developing service based systems/web services is BPEL/BPEL4WS (Business Process Execution Language for Web Services). A more detailed discussion of BPEL is provided in the Preliminaries section. While BPEL is a powerful tool for developing service based systems it lacks any formal semantics. It also does not provide any automatic service composition and adaptation rules. BPEL is great for modeling and creating workflows, its XML based schema is fairly simple to use. But, the lack of formal semantics makes it a non provable language.

The other major language for creating services is OWL-S; it provides constructs for unambiguously describing the properties and capabilities of web services. OWL-S provides some limited formal guarantees, but again is not provable since there are no equivalence theories. Also, like BPEL OWL-S fails to provide any automatic service composition methods.

Some of the existing formal approaches for defining service based systems are rule based modeling (SWORD). In SWORD a service is represented by a rule that says that given certain inputs the service is able to produce a specific output. A rule based expert system is then used to automatically check to see if a composite service can be created using a set of existing services. Unfortunately, this method does not allow for services to have side effects. Also, no known work has been found that uses SWORD to create situation awareness or security policies.

Of course there are the classic process calculi like pi-calculus, ambient calculus, etc. While these do provide ways for formal reasoning, they do not provide any methods to process situation information and react to it. One could of course use linear logic, but it is non decidable and only provides semi automated service composition methods.

The collaborative project between Arizona State University and West Virginia University approach attempts to address these issues by providing a formal programming model for service based systems. It is based on the classical process calculus model, but includes operational semantics that involve interactions between: external actions – communication, leaving and joining groups, internal computations – method calls of named services. It can also model timeouts and failures in a monadic style and implement access control using hierarchical groups.

In the next few sections we will gain a better understanding of what service based systems/web services are. Following that, we will take a look at the languages α-calculus and SOL, as well as SINS – a middleware for agents in service based systems. Finally, a brief introduction to Gentle, the compiler construction tool used to create the α-calculus to SOL compiler, will be provided.

PRELEMINARIES

What are service-based systems?

Service-based systems are becoming more and more prevalent as a way to create large scale distributed systems for applications ranging from mission critical to simple commercial. In a service based world applications are developed by the combination and coordination of different services. The individual services can reside in a single host server or be spread across the World Wide Web on multiple different servers. When a client makes a request for a specific task, a real time analysis is made of the task at hand to find out which services can be combined to fulfill the client's needs. Since it is possible that the requests made by the clients can be totally obtuse from one another, it is very important for the system to function in real time and be adaptive in nature.

Just like countries and states have governing bodies, service based systems too have their governing body called the Service Level Agreements (or SLAs). These also specify a certain Quality of Service (QoS) level that the service providers have to meet. There are essentially seven requirements in a given QoS agreement – availability, accessibility, integrity, performance, reliability, regulatory and security.

• Availability refers to the quality aspect of whether or not service is present or ready for immediate use. The score is given as a probability, a higher value means the service is always ready while a smaller value means the service may or may not be available. Also, a time-to-repair (TTR) value is assigned; this refers to how long it would take the service to get repaired if it failed. Naturally a smaller value is more desirable.

• Accessibility refers to the quality aspect of the service that deals with the degree to which it is capable of handling a service request. It can be measured as a probability ratio. A service can be available but not accessible; a high degree of accessibility can be achieved by using scalable systems.

• Integrity refers to the quality aspect of how well the service maintains the correctness of the interaction with respect to the source. Proper execution of a service transaction will provide this correctness of interaction. If a transaction does not complete then the changes that were made in the course of the transaction are rolled back.

• Performance refers to the quality aspect that measures how well a service is working. It is measured in terms of throughput and latency. Throughput is the number of service requests handled in a given period of time, and latency is the time between when a service request is made and a response is received.

• Reliability refers to the quality aspect of services that measures the ability of being able to maintain the service and its quality. The number of failures per day/month/year could be a measure of reliability. We could also think of reliability as the measure of whether a service requested matches the service provided.

• Regulatory refers to the quality aspect of services that measures its ability to comply with the rules, laws, standards, etc of the service level agreement. Adherence to standards is necessary in providing a good quality of service.

• Security refers to the quality aspect of services that measures its ability to provide confidentiality and non repudiation by authenticating the parties involved. Security is vitally important as a lot of these services interact over the Internet which is a very public domain.

Again, like any nation state the SLAs provide a penalty framework for dealing with infractions to the QoS agreement. Since service based systems span networks it is important to have these penalties to ensure that there are no security breaches, or failure to provide a service in a timely manner, etc. It is often very difficult for service providers to live up to the QoS because the network is always open to attacks.

Service-based systems are being increasingly used in mission critical applications such as the military, hospital, aviation, e-commerce, etc. Given the sensitivity of these applications the QoS needs to include adaptability, situation awareness and security. Adaptability refers to the system's ability to react to an external action while still providing the service; situation awareness refers to the ability to detect changes in the environment and adapt as needed. If a failure should occur the system should automatically find a service that can provide the same functionalities to the user and deploy it in place of the failed service.

A very simple service based system example could be an intrusion detection system. In this example we will have a Monitoring Agent, a Commander, several Ships and an Enemy. The Monitoring Agent detects an Enemy in a controlled zone, it sends out a signal to the Commander saying "Enemy Detected." The Commander in turn sends out a signal to the nearest ship, say Ship A, asking it to "Destroy Enemy". Ship A then executes the command, using the information provided. Naturally this is a rather simplistic example; we could enable some failsafe methods into it. Say Ship A does not respond in 10 seconds, the Commander can then call up Ship B with the same request and Ship B can neutralize the Enemy. We could even take it a step further and make it adaptive, the Commander could select Ships based on who is closet to the Enemy.

If we were to develop this system there would naturally be some security policies we would want to enforce. For instance, we only want the Commander to communicate with the Ships. The Monitoring Agent should not get direct access to the Ships, the reason being the Monitoring Agent could easily be compromised. Also, response time has to be critical since this is real time based. If Ship A doesn't respond there needs to be contingency plans to contact Ship B, Ship C, etc. Since the Enemy is moving there needs to be real time adaptability; if Ship A fails to respond then we can't use the same co-ordinates when asking Ship B to neutralize the Enemy because they may or may not be correct.

Figure 1: Intrusion Detection Service Based System

What are web services?

Web services are simple, self contained, modular, distributed and dynamic applications that can be used to perform a variety of functions. Web services allow machine-machine interaction as opposed to human-machine interactions. Web services are simply a specific type of service based systems, and hence are governed by the same Service Level Agreements and Quality of Service rules. An example of a web service would be orchestrating a pizza lunch at an office. First we would need to contact a secretary to obtain a free room, and then we would need to contact a pizza vendor and provide them with the room number and order request. A web service for this would automate everything: first by contacting the secretary's computer and obtaining a free room and booking it, then contacting the pizza vendor's computer and providing them with the room information and order information.

The concept of the World Wide Web is slowly changing from one of human interaction to one of machine interaction. In the forefront of this machine interaction movement are web services; they are being used more frequently now as we move towards the second generation of the World Wide Web. Web services are being deployed in mission critical applications like stock exchanges, hospital and health care, military, etc. Failure in these environments could have catastrophic consequences.

Web services have become increasingly popular in the recent years; the following are some major companies that provide a web service based infrastructure.

- Amazon.com - Search Products, Product Information, Cart System, Wish List
- eBay - Auction Search, Bidding, Auction Creation
- FedEx - Tracking
- Google - Web Search, Maps
- Mappoint - Maps
- MSN - Virtual Earth
- PayPal - Payment System
- Xignite - Financial market data
- Yahoo! - Maps, Traffic

In the next section we will look at a language called BPEL that is commonly used to develop web services. Despite being relatively new it has become an industry standard for developing web services.

What is BPEL

BPEL4WS (BPEL in short) – Business Process Execution Language for Web Services is one such language created for web services. BPEL4WS as formally defined by IBM is: "Business Process Execution Language for Web Services provides a means to formally specify business processes and interaction protocols. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces."

BPEL was designed to be what is called a large programming language, as opposed to a small programming language like C or C++. A few companies had large programming languages; like Microsoft had WSFL and IBM had XLANG. BPEL was developed by combining WSFL and XLANG into one language and in 2003 BEA System, IBM, Microsoft, SAP and Siebel Systems submitted BPEL 1.1 to OASIS for standardization.

The developers of BPEL came up with ten design goals. The follow list was taken from the Wikipedia page on BPEL.

- Goal 1: Define business processes that interact with external entities through Web Service operations defined using WSDL 1.1, and that manifest themselves as Web services defined using WSDL 1.1. The interactions are "abstract" in the sense that the dependence is on portType definitions, not on port definitions.

- Goal 2: Define business processes using an XML based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.

- Goal 3: Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

- Goal 4: Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.

- Goal 5: Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.

- Goal 6: Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.

- Goal 7: Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as "suspend" and "resume" may be added in future releases for enhanced lifecycle management.
- Goal 8: Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.
- Goal 9: Use Web Services as the model for process decomposition and assembly.
- Goal 10: Build on Web services standards (approved and proposed) as much as possible in a composable and modular manner.

BPEL not only provides the ability to send and receive messages it also supports the following as described on the Wikipedia site for BPEL.

- A property-based message correlation mechanism
- XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports Xpath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

As powerful as BPEL is in orchestrating web services, it lacks a formal semantics and cannot be verified. This leads to huge problems, how does a developer know their code actually does what it's supposed to? For instance, even the simplest Hello World program may not actually provide the correct output and instead may cause the system to crash and give undesired results. At first glance the problem seems unsolvable since BPEL is neither provable nor verifiable. But, what if there existed a language that was provable and another that was verifiable. One could easily design a translator to go from one to the other. Assuming, that the translator in itself is provable/verifiable we suddenly have one approach for creating provable/verifiable BPEL code.

In the next section we will look at one such novel language called α-calculus, a process calculus, which can be proven using mathematical induction. Following that we will look at SOL (Secure Operations Language), a verifiable synchronous language for developing reactive systems. Both of these languages in conjunction with their translators can enable us to create BPEL code that has already been proven and verified.

What is α-calculus?

α-calculus is a language developed in-house for the purpose of creating a formal approach for defining service based systems. A system defined in α-calculus can be a recursive process, or a system lying in a node enclosed in a firewall or the parallel combination of two systems. A node is defined as the space enclosed by the firewall with a name, and the name can be considered as the password used to access the node. A recursive process can be of several different types: it can be a name restriction, the inactive base process, parallel combination of two processes, an identifier for a process, a process executing an internal action, a process executing an internal computation, a time out process, a failed process or a service that exports methods.

We will now look at the syntax and grammar of α-calculus:

(System)

S::=

fix I=P	(recursion)
N[S]	(named domain)
S||S	(par. systems)

N ::=

x	(variable)
n	(name)

(Process)

P::=

(new n) P	(name restriction)
0	(inactive process)
P||P	(par. composition)
I	(identifier)
E.P	(external action)
C.P	(int. computation)
P1+P2	(nondet. choice)
fail(I)	(failure)
catch(I).P	(failure handler)
time t.P	(timeout)
P{l1(x1),…;…ln(xn)}	(method export)

(External Actions)

E ::=

M	(Domain)
K	(Comm.)

K::=

Ch(x)	(input)

	Ch<Str>	(output)
	mc(C1,…,Cn)<Str>	(multicast)

Ch::= N (Channel)

M ::=

	in N	(enter a dom.)
	out N	(exit a dom.)
	open N	(open firewall)
	ε	(no action)

(Internal Computations)

C::=

	let x=D instantiate C	(let reduction)
	if ρ then P else P’	(conditional)
	replace(I:li)	(method replacement)
	li ← lj	(method modification)
	ρ	(constraint evaluation)
	ε	(no-computation)
	tt	(constant true)
	ff	(constant false)
	⊥	(failed computation)

D::=

	I:li(y)	(method invocation for identified service)

I:li=

prei::posti[y]

pre::= σ[y] ∧ ρ[y]

post::= (σ[x] ∧ ρ[x]) x

(Types)

σ::=

	b	(base type)
	σ→σ	(function type)

(Linear arithmetic constraints)

x,y	(variable)
c	(natural number)

ρ::= (constraint)

x>c

x<c

x>=c
x=<c
x==c
x>y
x<y
x=<y
x>=y
x==y
x >=y+c
x>y+c
x =< y+c
x<y+c
x==y+c
ρ&&ρ

Back in the Preliminaries section we talked about service based system and described an intrusion detection system as an example, now we will look at how we can define the same system using α-calculus. Our agents are the Monitoring Agent, the Commander and the Ships.

Monitoring Agent:

fix MA= let z=MA:detect_intrusion() instantiate if z=true then let <x,y>=MA: get_enemy_coordinates() instantiate ch<x,y>.MA else MA

Commander:

fix CMD=ch(x,y). ch1<x,y>.<destroy>.CMD

Ships:

fix shipA= ch1(x,y).ch2(d). if d="destroy" then let t= (shipA:lock_radar(x,y) instantiate let s=instantiate shipA:load_missile().(let z=shipA:fire(x,y) instantiate if z= enemy_destroyed then <z>) then shipA) else shipA

What makes α-calculus novel is that we can easily prove the above system and ensure that the Ship eventually executes the destroy command. In order to do this proof we first need to have some basic assumptions:

- Every service will eventually return
- If an enemy is detected detect_intrusion will return a value
- get_enemy_coordinates will return a value once detec_enemy returns a value
- lock_radar will return a value once get_enemy_coordinates has passed the values to it
- load_missile will return true once lock_radar has been evaluated
- enemy_destroyed will return a value

Each of the 5 functions will have some pre and post conditions as follows:

detect_intrusion

pre condition – true
post condition – true or false

get_enemy_coordinates

pre conditions – true
post conditions – x: x-coordinate, y: y-coordinate

lock_radar

pre conditions – x: x-coordinate, y: y-coordinate
post conditions – true

load_missile

pre condition – x: x-coordinate, y: y-coordinate
post condition – true

enemy_destroyed

pre condition – x: x-coordinate, y: y-coordinate
post condition – enemy destroyed or enemy not destroyed

Now, we can take a look at the operational semantics of the language before going into the actual proof. We will use six simple rules to derive the proof system, they are as follows:

$$\frac{ch\langle t\rangle.P \parallel ch(x).Q,\ x{:}T_1}{P \parallel Q\ [t/x],\ t{:}\ T_1} \quad (1)$$

$$\frac{I{:}l_i \rightarrow pre{:}post,\ y_i \models pre}{I{:}l_i \rightarrow post} \quad (2)$$

$$\frac{\rho \rightarrow true}{\text{if } \rho \text{ then P else P'} \rightarrow P} \quad (3)$$

$$\frac{\rho \rightarrow false}{\text{if } \rho \text{ then P else P'} \rightarrow P'} \quad (4)$$

$$\frac{A \rightarrow B}{A.B \rightarrow B.P} \quad (5)$$

$$\frac{X \rightarrow t}{\text{let } z=X \text{ instantiate } P \rightarrow P[t/x]} \quad (6)$$

Now, if an enemy is detected then the post condition of detect_intrustion will be true. This will result in get_enemy_coordinates settings its post condition to the coordinates of the enemy. After that the system will instantiate ch<x,y> which contains the enemy coordinates and pass it to the commander. The commander in turn will pass the command to destroy to the ships. The ships will first lock_radar on the enemy using the coordinates from the commander and then will load_missile with the same coordinates before invoking the fire command that will destroy the enemy. Thus we can see from the command detect_intrusion we eventually reach the command enemy_destroyed, thereby proving that the system actually does what it is supposed to do.

In the following section we will begin looking at Secure Operations Language (SOL) which is one of the many languages available for developing service based systems. One of the great aspects of SOL is that it is verifiable thus providing the developer with the satisfaction of knowing they can actually verify the code written is adhering to its requirements.

What is Secure Operations Language (SOL)?

SOL is a synchronous programming language, developed at the Naval Research Labs, which can be used to develop reactive systems. The language finds its place in application development in mission critical cases like the navy, military, medical industry, etc. The need for a language like SOL arose with the desire to create high assurance systems; that is a system that provides security, safety, timeliness, survivability and fault tolerance. One key aspect of SOL that sets it aside from other more contemporary languages is that it is a secure language, which means it is open to automatic theorem proving or model checking.

SOL is an event driven language with system behavior modeled as modules. Each module may be either deterministic, reactive or both. Each module also contains state variables falling into the following categories: monitored, controlled or internal. As the names suggest; monitored are environment variables that are watched by the module, controlled are those variables that the module controls and internal variables are updated by the module but not visible to the environment. Each controlled and internal variable of the module has one definition that will determine when and how the variable gets updated. The assumptions section includes any assumptions about the environment the user is making; it is up to the user to ensure there are no logical contradictions. The guarantees section contains what the invariants are. The definitions section is the core part of a SOL module and contains variable interactions just like in any other programming language.

The basic code structure in SOL is as follows:

```
module module_name {

type definitions
        …
monitored variables
        …
controlled variables
        …
internal variables
        …
assumptions
        …
guarantees
        …
definitions
        …
} // end module_name
```

The figure below, from *SOL: A Verifiable Synchronous Language for Reactive Systems by Dr. Ramesh Bharadwaj*, shows the available syntax for SOL definitions.

defn : *lvalue* "=" *expr* | *lvalue* "=" "initially" *expr* "then" *expr* ";"

lvalue : *ID* | *ID* "[" *index* "]" | "[" *lvalue* ["," *lvalue*]* "]"

expr : *value* | "!" *expr* | *expr bool_binop expr* | *if_expr* | *case_expr* | *basic_event* | *cond_event* | "PREV" "(" *expr* ")" | *expr rel_binop expr* | "+" *expr* | "-" *expr* | *expr arith_binop expr* | *ID* "[" *index* "]" | *ID* "(" [*expr_l*]? ")" | "[" *expr_l* "]" | "(" *expr* ")"

if_expr : "if" "{" ["[]" *expr* "->" *expr*]+ ["otherwise" "->" *expr*]? "}"

case_expr : "case" *expr* "{" ["[]" *value* ["," *value*]* "->" *expr*]+ ["otherwise" "->" *expr*]? "}"

cond_event : *basic_event* "when" *expr*

basic_event : "@*ID* " ["(" *expr_l* ")"]? | "@T" "(" *expr* ")" | "@F" "(" *expr* ")" | "@C" "(" *expr* ")"

expr_l : *expr* ["," *expr*]*

value : *index* | *REAL* | *STRING* | "true" | "false" | "infinity"

index : *scalar_value* | *scalar_value* ":" *scalar_value*

scalar_value : *ID* | *INT*

bool_binop : "&" | "&&" | "|" | "||" | "=>" | "<=>"

rel_binop : "<" | "<=" | "==" | "!=" | ">" | ">="

arith_binop : "+" | "-" | "*" | "/"

Legend:

| Choice
[ ]? Optional
[ ]* Zero or more
[ ]+ One or more

Figure 2: Syntax for SOL Definitions

Verification of SOL can be performed by a tool called SALSA – an invariant checker for reactive synchronous systems. If an invariant is not provable, SALSA will return a counter example that the user must verify to ensure it is actually reachable. SALSA not only provides invariant checking but also consistency checking. The latter is very helpful as it returns errors if there missing cases in a modules or undesirable non determinism instances.

While SOL allows a developer to create agents for reactive systems, a middleware is needed to ensure there is a certain degree of trust between the mobile agents and also ensuring that they comply with certain security policies. In the next section we will discuss one such middleware called SINS (Secure Infrastructure for Networked Systems).

What is Secure Infrastructure for Networked Systems (SINS)?

How does one develop a trusted application that resides in a distributed network like the Internet? The Internet is open and prone to attacks ranging from Trojans, viruses, to distributed denial of service attacks. SINS, developed at the Naval Research Labs, is one way to ensure mobile agents have the required degree of trust. SINS is a middleware for secure agents and not only provides the required degree of trust for the mobile agents, but also ensures they are compliant with a set of enforceable security policies.

As outlined in the paper *SINS: A Middleware for Autonomous Agents and Secure Code Mobility by Dr. Ramesh Bharadwaj*, SINS addresses the following requirements of secure mobile agents:

- The author and initiator of an agent must be authenticated.
- The integrity of an agent's code must be checked.
- Interpreters must ensure that agent privacy is maintained during data exchange.
- Interpreters must protect themselves against malicious agents.
- Interpreters must ensure that migrating agents are in a safe state.
- Agents must protect themselves from malicious hosts and interpreters.
- An initiator must be able to control an agent's flexibility; i.e., restrict or increase an agent's authorization in specific situations.
- Initiators must be able to control which interpreters are allowed to execute their agents.

As discussed in the previous sections the mobile agents are created using SOL. As explained before, an application written in SOL is comprised of a set of modules. To ensure the ease of application creation the code is created using Visual SOL, much like Visual C or Visual BASIC this ensures a faster and easier application development. Each of these modules is run on what is called an Agent Interpreter. The Agent Interpreter then executes the modules based on a certain set of security policies that have been defined locally. SOL applications are not limited to a single Agent Interpreter; they could run on multiple Agent Interpreters spread over a vast network. This makes the problem of enforcing the local policies; the application could be within the guidelines of one policy while beyond the scope of another. Thus, care has to be taken when writing the applications to ensure they follow each local security policy.

There are a few basic assumptions about what SINS can and can’t do, they are as follows:

- Each agent interpreter will be run to completion by the hosts.
- Each agent will be run correctly by the agent interpreters; naturally if the interpreter doesn’t run them correctly we can’t say for certain whether or not they are following the security policies, etc.
- Each agent interpreter will transfer agent data as requested; again we hope the interpreter doesn’t randomly transfer data when it wasn’t asked to.
- The agent code and data will not be kept private from the host that is running the application.

- Intercommunication between agents will not be kept private from hosts. We don't want malicious agents to compromise the host that is running the application.
- Agent cannot carry any secret keys; given that the host could be compromised we don't want any secret keys to be leaked out.

SINS also attempts to address the following security issues faced by mobile agents.

- **Authentication and authorization** – in order to prevent the agent code from getting tampered by malicious hosts, or hosts that have somehow become compromised the SOL code is digitally signed and stored in a code repository. When handling a mobile agent each host will retrieve the code either from the repository or from another host that has the most recent copy. SINS also provides role based access control and management (RBAC) and trust management (TM) utilities to help authenticate the agent initiator, and to provide access to host resources.
- **Integrity of agent code** – since the agents are programmed in SOL, a verifiable language, one can gain a certain degree of trust knowing that the programs will have no unbounded loops, no buffer overflows, etc. Other languages like Java, C, C++ are non verifiable, hence a client has to trust that the developer hasn't inadvertently caused any problems. SINS proves each agent is in compliance with the local security policies by using its compliance checker.
- **Agent privacy** – agent activities are monitored by SINS using its own security architecture. Also, Agent Interpreters intercommunicate using SSL to prevent eavesdroppers from gaining access to private information.
- **Protection from malicious agents** – malicious agents could not only compromise the host that is running them, but also other agents and applications. In order to protect the host from malicious agents the compliance checker can detect any malicious activity.
- **Safe agent migration** – a migrating agent could easily become malicious and in order to prevent this from happening, each agent has what is called a state appraisal function. The state appraisal function simply checks to make sure that the agent will perform as required and the data hasn't been tampered with. The compliance checker can also be used to ensure the appraisal function does what it is supposed to do and meets certain safety requirements.
- **Agent protection from malicious hosts** – while one has to protect the host from malicious agents, the converse also has to be done in order to prevent the host from obtaining classified data from the agents. Currently there is no real framework to help protect agents from hosts, but a sort of protection force is set up using security agents. These, much like their human counterparts, ensure that the individual agents don't partake in any malicious activities.

In the next section we will take a brief look at Gentle, the compiler construction tool used to develop the α-calculus to SOL compiler before moving onto an in depth look at the actual compiler.

An introduction to Gentle

Compiler construction in the past has been arduous because of the lack of a good compiler construction tool. Programming even the simplest of compilers using a low-level language like C has been time consuming and often error prone. Today compiler construction, although not any simpler, is made easier with the availability of tools like Gentle, Antlr, etc. Gentle allows for a very high level definition of a compiler and avoids some of the pitfalls of more standard compiler construction practices.

A simple example of a calculator that takes a set of arithmetic functions and prints them when written in Gentle would look something like this.

```
'root' expression(-> X) print(X)

'nonterm' expression(-> INT)
        'rule' expression(-> X): expr2(-> X)
        'rule' expression(-> X+Y): expression(-> X) "+" expr2(-> Y)
        'rule' expression(-> X-Y): expression(-> X) "-" expr2(-> Y)

'nonterm' expr2(-> INT)
        'rule' expr2(-> X): expr3(-> X)
        'rule' expr2(-> X*Y): expr2(-> X) "*" expr3(-> Y)
        'rule' expr2(-> X/Y): expr2(-> X) "/" expr3(-> Y)

'nonterm' expr3(-> INT)
        'rule' expr3(-> X): Number(-> X)
        'rule' expr3(-> - X): "-" expr3(-> X)
        'rule' expr3(-> + X): "+" expr3(-> X)
        'rule' expr3(-> X): "(" expression(-> X) ")"

'token' Number(-> INT)

'root' expression(-> X)
        code(X)

'action' code (Expr)

'rule' code(plus(X1, X2)):
            code(X1) code(X2) print("plus")
'rule' code(minus(X1, X2)):
            code(X1) code(X2) print("minus")
'rule' code(mult(X1, X2)):
            code(X1) code(X2) print("mult")
'rule' code(div(X1, X2)):
            code(X1) code(X2) print("div")
'rule' code(neg(X)):
            code(X) print("neg")
'rule' code(num(N)):
            print(N)
```

α-CALCULUS TO SOL TRANSLATOR

Background

In developing a formal approach for developing service based systems, my contribution was developing the α-calculus - SOL translator. The translator is a key element in developing the formal approach, one first writes code in α-calculus and then proves that the code actually holds true with the given requirements. After this the code is passed through the translator and the resultant SOL code can be verified to once more ensure that it is in compliance with our requirements before being deployed. Of course one basic assumption is that the translator in itself in provable; while this task hasn't been completed yet it can be done fairly easily.

Having worked on the initial development of the SOL – BPEL (a discussion of the SOL – BPEL translator will come later) I realized that writing the translator in C would not be a good idea. I needed a compiler/translator construction tool that was fairly easy to use and the learning curve wasn't very steep. I came up with two options – Gentle or Antlr. Both of these have been heavily used in industry with a great deal of success. I ended up choosing Gentle because its syntax was very similar to Prolog, a language that I had worked with before. A few weeks of writing simple code in Gentle and reading the manual and I had a rudimentary understanding of how to use it to create the translator.

Given the complexity of the translator a good testing mechanism was needed, past projects were simple enough that I could write the whole code and then perform the required tests. Doing that for the α-calculus – SOL translator would be disastrous, hence a modular design structure was chosen. First and foremost I needed to have test cases, ranging from the very basic to the complex. Each test case then had to be translated by hand and verified to ensure I had a model to check the machine generated translation. The translator was built in parts, starting of with simple "Put" statements for some of basic SOL elements like "monitored variable", "controlled variables", etc. As layers of complexity were added to include conditionals, loops, etc they were tested to ensure nothing prior had been affected. This ensured the debugging process did not get too out of hand.

In order for the translator to work correctly, it has to first parse the α-calculus code and ensure it is correct. Then it will use the tokens that are created in the parsing phase to translate the code into SOL. It is critical to get the parsing done correctly, as without the proper tokens the translated SOL code will not be correct. The next section will provide a directory structure, installation instructions and a short guide on how to use the translator.

A short guide to using the translator

This software is designed to run in any version of Unix/Linux. The Gentle Compiler Construction Tool is needed in order for this compiler to work. The Gentle Compiler Construction Tool can be downloaded at http://gentle.compilertools.net/ or by following the direct link: http://gentle.compilertools.net/gentle-97.tar.gz. Installation of Gentle is achieved by executing the following command from the command line:

gunzip gentle-97.tar.gz
tar -xvf gentle-97.tar

This creates a directory gentle-97 with subdirectories gentle, lib, reflex, and examples. Go to directory gentle and type:

./build

This creates the executable program gentle, the Gentle compiler, as well as the object file *grts.o*, the Gentle run time system. Go to directory lib and type

./build

This compiles the modules of the user library. Go to directory reflex and type

./build

This creates the executable program reflex, a generator for Lex specifications.

A pre built software package will consists of the following files. The file *build* is used to build the translator, the file *trans.g* is the translator written in Gentle and *trans* is the executable file.

build
gen.h
gen.l
gen.lit
gen.tkn
gen.y
Ident.t
lex.yy.c
lex.yy.o
Number.t
prn.c
prn.o
String.t
trans
trans.c
trans.g
trans.o

y.tab.c
y.tab.o

If the software package hasn't been built yet or is missing some files then executing *./build* will build it and restore the files. The file *trans* is the executable that will be used to run the translator.

Once the executable has been created the user must create sample test files containing code written in α-calculus. A file extension is not required for the test files, but the test files must reside in the directory containing the *trans* executable. Say, the user creates a test file called *test* and wants to convert it to SOL. The conversion process is invoked by typing *./trans test* from the command line. If the input file is syntactically correct the converted SOL file will be created and stored in *out.sol*.

In the next section we will begin looking at the translator is further detail, first by looking at how the input test file is parsed and then looking at a fully worked example before moving onto how the translator actually translates the files.

Parsing rules

To understand how the translator converts from α-calculus to SOL one has to look at how the code is parsed. Successful parsing is critical for the translator to work; only if the code is parsed correctly will it get translated into SOL. First we will look at each non terminal rule and get a better understanding of what code sections it handles and the tokens that are created. The parser is invoked by calling Program(->P), only upon successful return is the translator called.

Program:

- A program of type SYSTEM returns a system of type SYSTEM like **fix A = zero** will return **fix A = zero** as a system.

System:

- A system can be a parallel combination of two systems like **fix A = zero || fix B = zero**, where **fix A = zero** and **fix B = zero** are both of type SYSTEM
- A system can be of the type **fix A = zero**, where A is an identifier of type IDENT and **zero** is a process of type PROCESS
- A system can be of the type **fix fail A = zero**, where A is an identifier of type IDENT and **zero** is a process of type PROCESS

Process:

- A process can be a parallel combination of two processes like **zero par zero**, where **zero** is of type PROCESS
- A process can be a combination of an external action and a process like **integer A(integer B).zero**, where **integer A(integer B)** is an EXT_ACT and **zero** is a PROCESS
- A process can be a combination of an internal computation and a process like **if A>5 then zero else zero.zero**, where **if A>5 then zero else zero** is an INT_COMP and **zero** is a PROCESS
- A process can be an addition of two processes like **zero pl zero**, where **zero** is of type PROCESS
- A process can be of the type **catch (A).zero**, where **A** if of the type IDENT and **zero** is of the type PROCESS
- A process can be of the type **failref (A)**, where **A** is of the type IDENT
- A process can be of the type **timeout 0.zero**, where **0** is if type INT and **zero** is of type PROCESS
- A process can be simply **A**, where **A** is of type IDENT
- A process can be **zero**

ExtAct:

- An external action can be **in name A**, where **A** is of type IDENT
- An external action can be **name A**, where **A** is of type IDENT
- An external action can be **out name A**, where **A** is of the type IDENT
- An external action can be **out A**, where **A** is of the type IDENT

- An external action can be **open name A**, where **A** is of the type IDENT
- An external action can be **open A**, where **A** is of the type IDENT
- An external action can be **integer A(integer B)**, where **integer A** is of type CHANNEL and **integer B** is of type VAR
- An external action can be **integer A<integer B>**, where **integer A** is of type CHANNEL and **integer B** is of type VAR
- An external action can be **integer A<A>**, where **integer A** is of type CHANNEL and **A** is of type IDENT
- An external action can be **integer A<5>**, where **integer A** is of type CHANNEL and **5** is of type INT

IntComp:

- An internal computation can be **let integer r=integer S:Fire(real x, real y) instantiate integer c<integer r>**, where **integer** is of TYPE, **r** is of type IDENT, **integer** is of TYPE, **S** is of type IDENT, **FIRE** is of type IDENT, **real x**, **real y** are of type VAR_LIST, **integer x<integer r>** is of type GEN_ACT
- An internal computation can be **let integer r=integer S:Fire() instantiate integer c<integer r>**, where **integer** is of TYPE, **r** is of type IDENT, **integer** is of TYPE, **S** is of type IDENT, **FIRE** is of type IDENT, **integer x<integer r>** is of type GEN_ACT
- An internal computation can be if **aStatus>=0 then P else zero**, where **aStatus>=0** is of type CONSTRAINT, **P** is of type PROCESS, **zero** is of type PROCESS
- An internal computation can be **integer A:B<-integer C:D**, where **integer** is TYPE, **A** is of type IDENT, **integer** is TYPE, **B** is of type IDENT, **C** is of type IDENT, **D** is of type IDENT

GenAct:

- A general action can be an external action like **in name A**, where **in name A** is of type EXT_ACT
- A general action can be a internal computation like **if A>5 then zero else zero**, where **if A>5 then zero else zero** is of type INT_COMP
- A general action can be a combination of general actions like **if A>5 then zero else zero dot if B>5 then zero else zero**, where **if A>5 then zero else zero** is of type GEN_ACT and **if B>5 then zero else zero** is of type GEN_ACT

Channel:

- A channel can be **integer A**, where **integer A** is of type VAR

Varlist:

- A variable list can be a single variable **A**, followed by a list of variables **B,C,D**
- A variable list can be a single variable **A)**, that is a variable followed by a brace

Constraint:

- A constraint can be **A > 5**, where **A** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A < 5**, where **A** is an UNTYPED_VAR and **5** is an INT

- A constraint can be **A >= 5**, where **A** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A =< 5**, where **A** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A == 5**, where **A** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A > B**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR
- A constraint can be **A < B**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR
- A constraint can be **A >= B**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR
- A constraint can be **A =< B**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR
- A constraint can be **A == B**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR
- A constraint can be **A > B + 5**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A < B + 5**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A >= B + 5**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A =< B + 5**, where **A** is an UNTYPED_VAR and **B** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A == B + 5**, where A is an UNTYPED_VAR and **B** is an UNTYPED_VAR and **5** is an INT
- A constraint can be **A>B && B<C**, where **A>B** is of type CONSTRAINT and **B<C** is of type CONSTRAINT

Untyped_Var:

- An untyped variable can be **'A'**, where **A** is of type IDENT
- An untyped variable can be **' '** an empty string
- An untyped variable can be **A**, where **A** is of type IDENT

Var:

- A variable can be **integer A**, where **integer** is of TYPE and **A** is type IDENT

Type:

- A type can be either **integer, real, Name, bool, aldef** or assignment like **integer -> integer**

Time1:

- A time out is an integer number, like **5**

Parsing example

Below is a sample α-calculus code that we will run through the parser to see if it correctly parses. Once it parses we will continue into the translation phase which is discussed in the next section.

fix Q=integer c(integer r).zero

1. The Program first calls System.

2. System calls the rule System(->atsys1(I,P)) with **fix Q=integer c(integer r).zero**

3. **Q** is IDENT and **integer c(integer r).zero** is PROCESS

4. The Process rule Process(->dotext(E,P)) is called with **integer c(integer r)** as EXT_ACT and **zero** as process

5. The Ext_act rule comm(bracket(C,V))) is called with **integer c** as CHANNEL and **integer r** as VAR

6. The Channel rule Channel(->channel(C)) is called with **integer c** as VAR

7. The Var rule Var(->var(T,X)) is called with **integer** as Type, the type rule Type(->type(T)) checks to make sure **integer** is a valid type.

8. The Var rule Var(->var(T,X)) is called with **integer r**, which in turn calls Type with the rule Type(->type(T)) checks to make sure **integer** is a valid type.

9. The example has parsed out correctly and follows the grammar rules. Now we can translate this using the action rules specified.

Translation

The translation rules are fairly simple; once the original α-calculus code has been successfully parsed the translator simply translates each token into its component SOL code. The biggest problem comes when you have nested loops; in this case the translator has to follow each branch until it reaches a base case before backtracking to the root. Stacks are used to keep track of which loop the translator is in and how deep into a specific branch the translator is so that when it backtracks it reaches the correct end point.

After the parser has been invoked by calling Program(->P) and has successfully returned, the translator starts of by first setting stack and newstack to 0. Then the output file called out.sol is created and Pretranslate(P) is called. After successful translation the output file out.sol is closed and the program exits.

Pretranslate(P) calls Translate(P), if the input file consists of a parallel combination of two systems then it calls Translate(P) with each otherwise it creates a basic SOL skeleton code that includes the following type definitions: tstring=string and Name=string. (In the event of a failure type input file Name=string is left out from the skeleton code.) Also, dummy variables are included in the monitored, controlled and internal variables sections to be used if no variables are declared. After this it calls the individual functions to collect each variable type before calling TranslateProcess(P) which then translate the α-calculus code to SOL.

Each collection function first traverses the file to find each instance of the type of variable. Once this is done it prints them out in the corresponding section in the SOL file. The translator is designed so that even if the monitored, controlled and internal variables are called the same thing it will still assign them correctly.

TranslateProcess(P) either calls itself it's a parallel combination of two processes or depending on whether its an external action or an internal computation it calls the corresponding TranslateExt or TranslateInt respectively. TranslateExt and TranslateInt in turn calls different print functions to convert the α-calculus code to SOL.

The cross reference guide provided in the next section provides a clear outline of how the translator creates its tree. When a function is invoked it will execute it and all sub functions before returning.

Function cross reference table

The following table provides an easy function cross reference guide for each function and what it invokes. The table also helps in understanding the recursive nature of the program.

Function	**Invokes**
Pretranslate	Translate
Translate	Translate
	id_to_string
	Put
	CollectMonitoredVariable
	CollectControlledVariable
	CollectInternalVariable
	TranslateProcess
TranslateProcess	TranslateExt
	TranslateProcess
	Put
	IsStack
	TranslateInt
CollectMonitoredVariable	CollectMonitoredExt
	CollectMonitoredVariable
	CollectMonitoredInt
CollectMonitoredExt	checkdeclaredvariable
	PrintChannel
CollectMonitoredInt	CollectMonitoredGen
	CollectMonitoredVariable
	Put
	id_to_string
CollectMonitoredGen	CollectMonitoredInt
	CollectMonitoredExt
	CollectMonitoredGen
PrintVar	id_to_string
	Put
CollectInternalVariable	CollectInternalExt
	CollectInternalVariable
	CollectInternalInt
CollectInternalExt	PrintVar
CollectInternalInt	CollectInternalGen
	CollectInternalVariable
CollectInternalGen	CollectInternalExt
	CollectInternalInt

	CollectInternalGen
CollectControlledVariable	CollectControlledExt
	CollectControlledVariable
	CollectControlledInt
CollectControlledExt	checkdeclaredvariable
	PrintChannel
CollectControlledInt	CollectControlledGen
	CollectControlledVariable
CollectControlledGen	CollectControlledExt
	CollectControlledInt
	CollectControlledGen
TranslateExt	InsertVar
	InsertatC
	Put
	PrintVarEq
	ChecknotVar
	PrintVarChanEq
	PrintAllDefinitionsfromRest
	PrintChanConstantEq
	PrintChanIntEq
PrintAllDefinitionsfromRest	PrintAllDefinitionsfromRestExt
	PrintAllDefinitionsfromRest
	PrintAllDefinitionsfromRestInt
PrintAllDefinitionsfromRestExt	eq
	Put
	id_to_string
	InsertatCV
	PrintvariableNoType
	Insertattrue
PrintAllDefinitionsfromRestExtIf	eq
	Put
	InsertatCV
	id_to_string
	PrintvariableNoType
	Insertattrue
PrintAllDefinitionsfromRestExtOtherwise	eq
	Put
	InsertatCV
	PrintvariableNoType
	id_to_string
	Insertattrue

PrintAllDefinitionsfromRestInt	PrintAllDefinitionsfromRestInt
	PrintAllDefinitionsfromRest
	PrintAllDefinitionsfromRestExt
	PrintAllDefinitionsfromRestif
	PrintAllDefinitionsfromRestOtherwise
PrintAllDefinitionsfromRestif	Printcons
	Put
	PushConstraintIfVar
	PrintAllDefinitionsfromRestExtIf
	PrintAllDefinitionsfromRest
	ContainsOtherInstancesOfDefinition
	PrintAllDefinitionsfromRestInt
PrintAllDefinitionsfromRestOtherwise	ContainsOtherInstancesOfDefinition
	Put
	eq
	Printcons
	PrintAllDefinitionsfromRestExtOtherwise
	PrintAllDefinitionsfromRest
	PushConstraintOtherwiseVar
	PrintAllDefinitionsfromRestInt
InsertVar	id_to_string
	Put
	eq
InsertatC	id_to_string
	Put
	ne
InsertatCV	id_to_string
	Put
	ne
Insertattrue	Put
PrintVarEq	id_to_string
	Put
PrintVarChanEq	id_to_string
	eq
	Put
PrintVarChanEqOtherwise	id_to_string
	Put
	eq
	PutI
	explorestackif
	Printcons

	ne
PrintChanConstantEq	id_to_string
	Put
	eq
	ne
	explorestackif
PrintChanConstantEqOtherwise	id_to_string
	Put
	eq
	PutI
	ne
	explorestackif
	Printcons
PrintChanIntEq	id_to_string
	Put
	eq
PrintChanIntEqOtherwise	id_to_string
	Put
	eq
	PutI
	ne
	explorestackif
	Printcons
PrintChanConstantEqif	id_to_string
	Put
	eq
	PutI
	ne
	explorestackif
	Printcons
PrintChanIntEqif	id_to_string
	Put
	eq
	PutI
	ne
	explorestackif
	Printcons
PrintVarChanEqif	id_to_string
	Put
	eq
	PutI

	ne
	explorestackif
	Printcons
PrintC	id_to_string
	Put
atCvarlist	Put
	id_to_string
	ne
	PrintvariableNoType
	ContainsPrintableType
	atCvarlist
PrintRestServiceCalls	Put
	id_to_string
	atCvarlist
	PrintVarlist
	PrintRestServiceCalls
	PrintAllServicesFromRest
	PrintAllServicesFromRestIf
	PrintAllServicesFromRestOtherwise
PrintAllServicesFromRest	PrintAllServicesCalls
	PrintRestServiceCalls
PrintAllServicesFromRestIf	ContainsServices
	Put
	Printcons
	PrintAllServicesFromRest
PrintAllServicesFromRestOtherwise	ContainsServices
	Put
	Printcons
	PrintAllServicesFromRest
PushConstraintOtherwise	PushConstraintOtherwise
PushConstraintOtherwiseVar	PushConstraintOtherwiseVar
PushConstraintOtherwiseServ	PushConstraintOtherwiseServ
PushConstraintIf	PushConstraintIf
PushConstraintIfVar	n/a
PushConstraintIfServ	n/a
CloseBrackets	ifzero
	Put
	CloseBrackets
CloseBracketsVar	ifzero
	Put
	CloseBracketsVar

CloseBracketsServ	ifzero
	Put
	CloseBracketsServ
InsertVarIf	id_to_string
	Put
	eq
	PutI
InsertatCif	Put
	ne
	explorestackif
	Printcons
	id_to_string
explorestackif	explorestackifnext
explorestackifnext	eq
	Put
	ne
	Printcons
	explorestackifnext
explorestackifVar	eq
	ne
	Put
	Printcons
	explorestackifvar
explorestackifServ	eq
	ne
	Printcons
	explorestackifServ
TranslateInt	ifzero
	id_to_string
	Put
	atCvarlist
	PrintVarlist
	PrintRestServiceCalls
	TranslateGen
	PrintAllServicesFromRest
	TranslateProcessIf
TranslateProcessIf	TranslateOtherwiseProcess
	TranslateifModuleProcess
	TranslateExtIf
	TranslateProcessIf
	IsStack

	Put
	TranslateIntIf
	TranslateProcessIf
TranslateOtherwiseProcess	TranslateExtOtherwiseModule
	TranslateOtherwiseProcess
	IsStack
	Put
	TranslateIntOtherwiseModule
TranslateExtOtherwiseModule	InsertVar
	Put
	ne
	explorestackif
	Printcons
	PrintC
	PrintVarEq
	CloseBrackets
	ChecknotVar
	PrintVarChanEqOtherwise
	PrintAllDefinitionsfromRest
	PrintChanConstantEqOtherwise
	PrintChanIntEqOtherwise
TranslateIntOtherwiseModule	ifzero
	explorestackif
	Put
	Printcons
	atCvarlist
	id_to_string
	PrintVarlist
	PrintRestServiceCalls
	CloseBrackets
	PushConstraintOtherwise
	TranslateGen
	TranslateGenConsOth
	PrintAllServicesFromRest
	TranslateProcessIf
TranslateifModuleProcess	TranslateExtIf
	TranslateifModuleProcess
	TranslateIntIf
TranslateExtIf	InsertVarIf
	InsertatCif
	PrintVarEq

	Put
	CloseBrackets
	ChecknotVar
	PrintVarChanEqif
	PrintAllDefinitionfromRest
	ContainsOtherInstancesOfDefinitionCh
	TranslateExtOtherwise
	PrintChanConstantEqif
	PrintChanIntEqif
TranslateExtOth	InsertVarIf
	InsertatCif
	PrintVarEq
	Put
	CloseBrackets
	PrintVarChanEqif
	PrintAllDefinitionsfromRest
	ContainsOtherInstancesOfDefinitionCh
	TranslateExtOtherwise
	ChecknotVar
	PrintChanIntEqif
TranslateExtOtherwise	PrintAllDefinitionsfromRestExt
	PrintAllDefinitionsfromRest
	PrintAllDefinitionsfromRestInt
TranslateIntIf	ifzero
	ne
	explorestackif
	Printcons
	id_to_string
	atCvarlist
	PrintVarlist
	PrintRestServiceCalls
	ContainsServices
	TranslateOtherwiseInt
	CloseBrackets
	TranslateGenCons
	PrintAllServicesFromRest
	PushConstraintIf
	TranslateProcessIf
TranslateOtherwiseInt	TranslateOtherwiseInt
	TranslateOtherwiseIntRest
TranslateOtherwiseIntRest	Put

	id_to_string
	atCvarlist
	PrintVarlist
	PrintRestServiceCalls
	PrintAllServicesFromRest
	TranslateOtherwiseInt
TranslateGenCons	TranslateExtIf
	TranslateProcessIf
	TranslateIntIf
	TranslateGenCons
TranslateGenConsOth	TranslateExtOtherwiseModule
	TranslateProcessIf
	TranslateIntIf
	TranslateGenConsOth
TranslateGen	TranslateInt
	TranslateProcess
	TranslateExt
	TranslateGen
Printcons	PrintUntypedVariable
	Put
	Printnumber
	Printcons
Printnumber	PutI
PrintUntypedVariable	id_to_string
	Put
PrintVarlist	PrintvariableNoType
	Put
	ne
	PrintVarlist
Printvariable	id_to_string
	Put
PrintvariableNoType	id_to_string
	Put
PrintChannel	id_to_string
	Put
IsStack	eq
ifzero	eq
checkdeclaredvariable	eq
	notchannelsearch
	notchannelsearchlist
ChecknotVar	eq

	notsearch
	notsearchlist
notsearch	ne
notchannelsearch	ne
notsearchlist	notsearch
	notsearchlist
notchannelsearchlist	notchannelsearchlist
	notchannelsearch
ContainsServices	ContainsServices
	ContainsServicesInt
ContainsServicesInt	ContainsServices
ContainsOtherInstancesOfDefinition	ContainsDefinitionExt
	ContainsOtherInstancesOfDefinition
	ContainsDefinitionInt
ContainsDefinitionExt	eq
ContainsDefinitionInt	ContainsDefinitionInt
	ContainsOtherInstancesOfDefinition
	ContainsDefinitionExt
ContainsPrintableType	id_to_string
	ne
	ContainsPrintableType
ContainsOtherInstancesOfDefinitionCh	ContainsOtherInstancesOfDefinition
id_to_string	n/a

Translation example

Now that the section of code is correctly parsed we can begin the translation process. In this section we will see how it gets translated into SOL code.

fix Q=integer c(integer r).zero

1. First Pretranslate is called, which then calls Translate.

2. The rule Translate(atsys1(I,P)) is used, I = Q and P = integer c(integer r).zero

3. Translate then prints the following to the output file out.sol.

```
module Q{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;
```

   After this CollectMonitoredVariable(P) is called, where P = integer c(integer r).zero, upon returning from that the following is put into out.sol

```
controlled variables
integer dummyMonitored;
tString cServiceInvoke;
```

   Now the translator calls CollectControlledVariable(P), where P = integer c(integer r).zero to collect any controlled variables. Upon returning it prints the following to out.sol.

```
internal variables
integer dummyMonitored;
```

   Next we collect all internal variables by calling CollectInternalVariable(P), where P = integer c(integer r).zero. Upon returning the following is printed to the out.sol file.

```
definitions
```

   Next the translator calls TranslateProcess(P), again with P=integer c(integer r).zero. Once it returns from there it prints the following to the out.sol file.

```
}
```

4. Now we will look at CollectMonitoredVariable(P), where P=integer c(integer r). zero to see how it collects the variables. The rule CollectMonitoredVariable (dotext(E, P)) is called with E=integer c(integer r) and P=zero. This then calls CollectMonitoredExt(E), where E=integer c(integer r) and CollectMonitoredVariable(P), where P=zero.

5. First we look at CollectMonitoredExt(E), where E=integer c(integer r) to see what

it does. CollectMonitoredExt(E) uses CollectMonitoredExt(comm(bracket(C,V))) where C=integer c and V=integer r. This first checks to see C is already declared by calling checkdeclaredvariable(C), where C=integer c. After this it calls PrintChannel(C) with C=integer c.

6. PrintChannel(C), where C=integer c uses the rule PrintChannel(channel(var (type(T), X))) where T=integer and X=c. It prints the following to the file out.sol. (Note this section of code is placed right after integer dummyMonitored; in the file.)

```
integer c;
```

7. Now we got back to Step 4 and see how CollectMonitoredVariable(P), where P=zero is handled.

8. CollectMonitoredVariable(P) uses the rule CollectMonitoredVariable(zero) which does nothing. We have completed the collection of monitored variables. Now we go back and deal with the controlled variables.

9. CollectControlledVariable(P), where P=integer c(integer r).zero. The rule CollectControlledVariable (dotext(E, P)) is called with E=integer c(integer r) and P=zero. This then calls CollectControlledExt(E), where E=integer c(integer r) and CollectControlledVariable(P), where P=zero.

10. First we look at CollectControlledExt(E), where E=integer c(integer r) to see what it does. CollectControlledExt(E) uses CollectControlledExt(comm(bracket(C,V))) where C=integer c and V=integer r. This first checks to see C is already declared by calling checkdeclaredvariable(C), where C=integer c. Since it has already been declared we do nothing and the function exits.

11. Now we go back to Step 9 and see how CollectControlledVariable(P), where P=zero is handled.

12. CollectControlledVariable(P) uses the rule CollectControlledVariable(zero) which does nothing. We have completed the collection of controlled variables. Now we go back and deal with the internal variables.

13. CollectInternalVariable(P), where P=integer c(integer r).zero. The rule CollectInternalVariable (dotext(E, P)) is called with E=integer c(integer r) and P=zero. This then calls CollectInternalExt(E), where E=integer c(integer r) and CollectInternalVariable(P), where P=zero.

14. First we look at CollectInternalExt(E), where E=integer c(integer r) to see what it does. CollectInternalExt(E) uses CollectInternalExt(comm(bracket(C,V))) where C=integer c and V=integer r. This then calls PrintVar(V) where V=integer r.

15. PrintVar(V) where V=integer r uses the rule PrintVar(var(type(T),C)) with T=integer and C=r. It prints the following to out.sol, note this appears right after

integer dummyMonitored; in the file.

```
integer r;
```

16. So all variables have been declared and now we look at TranslateProcess(P) where P=integer c(integer r).zero to see how that gets converted to actual SOL code.

17. TranslateProcess(P) calls TranslateProcess(dotext(E,P)) where E=integer c(integer r) and P=zero. This then first calls TranslateExt(E,P) where E=integer c(interger r) and P=zero. One it returns from that it calls TranslateProcess(P) where P=zero.

18. TranslateExt(E,P) where E=integer c(integer r) and P=zero calls TranslateExt(comm.(bracket(C,V)),P) where C=integer c, V=integer r and P=zero. This then first calls InsertVar(V), where V=integer r. Upon returning it calls InsertatC(C), where C=integer c. Then it calls PrintVarEq(V,C) where V=integer r and C=integer c. Finally it prints the following to the file before exiting.

```
};
```

19. InsertVar(V) where V=integer r uses the rule InsertVar(var(type(T),X)), where T=integer and X=c. This prints the following to the file in the definitions section:

```
r= initially 0 then
```

20. InsertatC(C) where C=integer c uses the rule InsertatC(channel(var(type(T),X))), where T=integer and X=c. This prints the following to the file after r=initially 0 then

```
if {
[] @C(c) ->
```

21. PrintVarEq(V,C) where V=integer r and C=integer c uses the rule PrintVarEq (var(type(T),X),channel(var(type(T1),Y))) where T=integer, X=r, T1=integer and Y=c. This prints the following right after [] @C(c)->

```
c
```

22. Now we go back to Step 17 and look at TranslateProcess(P) where P=zero. This simply does nothing. We have now successfully translated the input file.

Thus the final translated SOL file looks like this:

```
module Q{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

integer c;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

internal variables
integer dummyMonitored;

integer r;

definitions

r= initially 0  then

if {
[] @C(c) -> c

};
}
```

Beyond SOL

While the α-calculus to SOL translator is a very powerful tool for creating reactive/service-based systems, it can be taken one step further with the use of the SOL to BPEL and SOL to Java translator.

The SOL to BPEL translator is currently under development, using the SOL to BPEL translator one could deploy provable BPEL code that could be used for designing web services. Given that BPEL in itself is neither provable nor verifiable one can first write code in α-calculus, prove that it does what it intends to by a simple proof. Then, using the α-calculus to SOL translator we can generate SOL code that is provable. Since SOL is verifiable, the code can then be verified before passing it through the SOL to BPEL translator.

The same principle can be used for the SOL to Java translator, while work has been completed on the SOL to Java translator a newer version is currently under development. Just like in the SOL to BPEL translator, we can first use the α-calculus to SOL translator to ensure we have provable code. Since Java is neither provable nor verifiable we can use the intermediary SOL code and ensure it is verified before passing it through the SOL to Java translator.

In the next section we will use the α-calculus to SOL translator along with the first generation SOL to Java translator to demonstrate how we can create provable/verifiable Java code. The one limitation of this method is naturally in order to modify the Java code we would have to modify the original α-calculus code to ensure that provability still exists. Whilst this may be a little redundant, it will allow the developer to have a sense of security knowing their modifications are still provable.

α-calculus to SOL to Java results

We will use the same sample code as used in the parsing/translation example.

The code written in α-calculus is as follows:

```
fix Q=integer c(integer r).zero
```

Passing it through the α-calculus to SOL translator we end up with:

```
module Q{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

integer c;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

internal variables
integer dummyMonitored;

integer r;

definitions

r= initially 0  then

if {
[] @C(c) ->

c

};
}
```

Passing the SOL code through the SOL to Java translator we get two files:

```
package testpackage;

import java.rmi.*;

public interface IQ{
  //*************************** accessor methods *******************************
  private void  setdummyMonitored(int value) throws Exception;
  private void  setc(int value) throws Exception;
  public int getdummyMonitored() throws RemoteException;
  public String getcServiceInvoke() throws RemoteException;
  public void reset() throws Exception;
}
```

```
import java.util.*;
import java.io.*;

import sins.SINSAgent;
import sins.SINSVariable;

public class Q extends SINSAgent {
  private INTEGER dummyMonitored = new INTEGER("monitored", "dummyMonitored");
  private INTEGER c = new INTEGER("monitored", "c");

  private INTEGER dummyMonitored = new INTEGER("controlled", "dummyMonitored");
  private tString cServiceInvoke = new tString("controlled", "cServiceInvoke");

  private INTEGER dummyMonitored = new INTEGER("internal", "dummyMonitored");
  private INTEGER r = new INTEGER("internal", "r");
  private Hashtable initMonVars=new Hashtable();
  private String justInitialized="false";
  private String timeStamp="0";

  public Q(){
          NAME = this.getClass().getName();
  }
  public Hashtable getVariables() {

   Hashtable htable = new Hashtable();

   htable.put("dummyMonitored", dummyMonitored.toString());
   htable.put("c", c.toString());
   htable.put("dummyMonitored", dummyMonitored.toString());
   htable.put("cServiceInvoke", cServiceInvoke.toString());

   return htable;
  }


  private void setAllFalse(){
   dummyMonitored.setHasBeenInitialized("false");
   c.setHasBeenInitialized("false");
   dummyMonitored.setHasBeenInitialized("false");
   cServiceInvoke.setHasBeenInitialized("false");
  }
  private boolean checkMonVars(){
   Vector MonVars = getMonVars();
   for (int i=0; i<MonVars.size(); i++)
   {
   if(!(initMonVars.containsKey(MonVars.elementAt(i)) ) )
   return false;
   }
   return true;
  }
  private Vector getMonVars(){
   Vector MonVars = new Vector();
   MonVars.add("dummyMonitored");
   MonVars.add("c");
   return MonVars;
  }
  public Hashtable evaluateMonitoredVariable(String varName, String value) throws Exception {
   Hashtable htable = new Hashtable();
```

```
  if(varName.equals("dummyMonitored")){
    dummyMonitored.setInit("false");
    dummyMonitored.setValue((Integer.valueOf(value)).intValue());
    htable = evaluate();
   }
   else if(varName.equals("c")){
    c.setInit("false");
    c.setValue((Integer.valueOf(value)).intValue());
    htable = evaluate();
   }
   else {
    System.out.println("Incorrect monitored Variable");
   }
   return htable;
  }
  public Hashtable handle_variable(String varName,String value,String init,String timeStamp )
throws Exception
  {
  Hashtable htable = new Hashtable();
  if(init.equals("true"))  /* handling initialization requests */
   {
  if(initMonVars.isEmpty())
   {
    setAllFalse();
   }
  else
   {
   SINSVariable dummy = (SINSVariable) initMonVars.get(varName);

   if(Long.parseLong(this.timeStamp) < Long.parseLong(timeStamp))
    {
     initMonVars.clear();
     setAllFalse();

    }
   }
  this.timeStamp = timeStamp;
  if(varName.equals("dummyMonitored")){
   dummyMonitored.setInit(init);
   dummyMonitored.setValue((Integer.valueOf(value)).intValue());
   dummyMonitored.setTimeStamp(timeStamp);
   dummyMonitored.clearTimeStamp();
   initMonVars.put("dummyMonitored",dummyMonitored);
  }
  else if(varName.equals("c")){
   c.setInit(init);
   c.setValue((Integer.valueOf(value)).intValue());
   c.setTimeStamp(timeStamp);
   c.clearTimeStamp();
   initMonVars.put("c",c);
  }
  else {
   System.out.println("Incorrect monitored Variable");
  }
      boolean allMonVarsSet = checkMonVars();
     if(allMonVarsSet)
      {
       justInitialized = "true";
```

```
        htable = evaluate();
        initMonVars.clear();
       }
     }
    else if(init.equals("false") && !initMonVars.isEmpty()) /*If update comes in the middle of
initialization */
    {
    /* currently discarding if any request for update comes in between */
    /* Please let us know if u want to handle these updates*/
    }
    else   /* To handle updates after initialization */
    {
     htable = evaluateMonitoredVariable(varName,value);
    }
    return htable;
     }
    /* (non-Javadoc)
     * @see container.services.ai.agent.AgentObject#evaluateInterface(java.lang.String)
     */
    public void evaluateInterface (String interfaceName) throws Exception {
    }

    /* (non-Javadoc)
     * @see container.services.ai.agent.AgentObject#getInterfaces()
     */
    public Hashtable getInterfaces () throws Exception {
     return null;
    }


    /* (non-Javadoc)
     * @see sins.agent.AgentObject#getControlledVariable()
     */
    public ArrayList getControlledVariables() throws Exception {

     ArrayList a1 = new ArrayList();
     a1.add(dummyMonitored);
     a1.add(cServiceInvoke);

     return a1;
    }

    /* (non-Javadoc)
     * @see sins.agent.AgentObject#getMonitoredVariable()
     */
    public ArrayList getMonitoredVariables() throws Exception {

     ArrayList a1 = new ArrayList();
     a1.add(dummyMonitored);
     a1.add(c);

     return a1;
    }

    /* (non-Javadoc)
     * @see sins.agent.AgentObject#getMonitoredVariable()
     */
    public ArrayList getInternalVariables() throws Exception {
```

```
 ArrayList a1 = new ArrayList();
 a1.add(dummyMonitored);
 a1.add(r);

 return a1;
}

/******************** START ACCESSOR METHODS**********************/
public void setDUMMYMONITORED(int value) throws Exception{
  dummyMonitored.setValue(value);
}

public void setC(int value) throws Exception{
  c.setValue(value);
}

public INTEGER getDUMMYMONITORED() throws Exception{
  return dummyMonitored;
}

public tString getCSERVICEINVOKE() throws Exception{
  return cServiceInvoke;
}

/******************** END ACCESSOR METHODS**********************/
public Hashtable evaluate() throws Exception {

 System.out.println("evaluate() ##1 called");
 boolean flag = false;

 if(justInitialized.equals("true")){
  r.setValue(0);
 }
 else {
  if(c.getChanged()||justInitialized.equals("true"))      {
  // ****************************
  flag=false;
  if((c.getChanged()))      {
  r.setValue(c.getVal());
  flag=true;
   }
   }
 }
 Hashtable htable = new Hashtable();
 if(justInitialized.equals("true")){
  dummyMonitored.setTimeStamp(this.timeStamp);
  dummyMonitored.clearTimeStamp();
  dummyMonitored.setInit(justInitialized);
  htable.put("dummyMonitored", dummyMonitored.toString());
  cServiceInvoke.setTimeStamp(this.timeStamp);
  cServiceInvoke.clearTimeStamp();
  cServiceInvoke.setInit(justInitialized);
  htable.put("cServiceInvoke", cServiceInvoke.toString());
 }
 else {
  if(dummyMonitored.getChanged())
  {
```

```
    dummyMonitored.setTimeStamp(this.timeStamp);
    dummyMonitored.increaseTimeStamp();
    dummyMonitored.setInit(justInitialized);
    htable.put("dummyMonitored", dummyMonitored.toString());
    }
    if(cServiceInvoke.getChanged())
    {
    cServiceInvoke.setTimeStamp(this.timeStamp);
    cServiceInvoke.increaseTimeStamp();
    cServiceInvoke.setInit(justInitialized);
    htable.put("cServiceInvoke", cServiceInvoke.toString());
    }
   }
   justInitialized ="false";
   dummyMonitored.setChanged(false);
   c.setChanged(false);
   dummyMonitored.setChanged(false);
   r.setChanged(false);
   dummyMonitored.setChanged(false);
   cServiceInvoke.setChanged(false);
   return htable;
  }


 public ArrayList getGroups() throws Exception {

   ArrayList a1 = new ArrayList();
   a1.add("dummyMonitored");
   a1.add("c");
   return a1;
  }
 /* Start of Type Definitions  */


 class Name implements SINSVariable{
   private String type = null;
   private String name = null;
   private String prev_name,name;
   private String init;
   private String hasBeenInitialized = "false";
   private long ts=0;
   boolean changed=false;

   public Name(String t,String n,String v){
    setType(t);
    setName(n);
    this.name = v;
    prev_name=name;
   }

   public Name(String t,String n){
    setType(t);
    setName(n);
    this.name = "";
    prev_name=name;
   }

   public String toString(){
```

```
  return String.valueOf(name);
}

public void setValue(String dummy){
  this.increaseTimeStamp();
  if(type.equals("monitored"))
  {
   if(init.equals("true"))
    {
     prev_name=dummy;
    name=dummy;
     hasBeenInitialized = "true";
    }
   else
    {
     prev_name=name;
    name=dummy;
    }
  }
  else
  {
        if(init == null||hasBeenInitialized.equals("false"))
        {
     prev_name=dummy;
    name=dummy;
      hasBeenInitialized = "true";
      init = "true";
        }
        else
        {
     prev_name=name;
    name=dummy;
        init = "false";
        }

  }
  changed = true;
}
public String getPrev(){
  return prev_name;
}

public void setPrev(String dummy){
  prev_name=dummy;
}

public String getName(){
  return name;
}

public String getVal(){
  return name;
}

public String getValue(){
  return String.valueOf(name);
}
```

```
  public String getPrevString(){
    return String.valueOf(prev_name);
  }

  public String getType(){
    return type;
  }

  public void setName(String n){
    name = n;
  }

  public void setType(String t){
    type = t;
  }

  public void setInit(String init){
     this.init = init;
  }
  public void setTimeStamp(String ts){
     this.ts = Long.parseLong(ts);
  }
  public long getTimeStamp(){
     return ts;
  }
  public void increaseTimeStamp(){
    this.ts ++;
  }
  public void clearTimeStamp(){
    this.ts = 0;
  }
  public String getInit(){
      return init;
  }
  public void setHasBeenInitialized(String hasBeenInitialized){
          this.hasBeenInitialized = hasBeenInitialized;
  }
  public String getHasBeenInitialized(){
    return            this.hasBeenInitialized;
  }
  public boolean getChanged(){
    return changed;
  }
  public void setChanged(boolean flag){
    changed =flag;
  }
}
class tString implements SINSVariable{
  private String type = null;
  private String name = null;
  private String prev_tstring,tstring;
  private String init;
  private String hasBeenInitialized = "false";
  private long ts=0;
  boolean changed=false;

  public tString(String t,String n,String v){
    setType(t);
```

```
  setName(n);
  this.tstring = v;
  prev_tstring=tstring;
 }

public tString(String t,String n){
  setType(t);
  setName(n);
  this.tstring = "";
  prev_tstring=tstring;
 }

public String toString(){
  return String.valueOf(tstring);
 }

public void setValue(String dummy){
  this.increaseTimeStamp();
  if(type.equals("monitored"))
  {
   if(init.equals("true"))
    {
     prev_tstring=dummy;
    tstring=dummy;
     hasBeenInitialized = "true";
    }
   else
    {
     prev_tstring=tstring;
    tstring=dummy;
    }
  }
  else
  {
         if(init == null||hasBeenInitialized.equals("false"))
         {
     prev_tstring=dummy;
    tstring=dummy;
       hasBeenInitialized = "true";
       init = "true";
          }
         else
         {
     prev_tstring=tstring;
    tstring=dummy;
         init = "false";
         }

  }
  changed = true;
 }
public String getPrev(){
  return prev_tstring;
 }

public void setPrev(String dummy){
  prev_tstring=dummy;
 }
```

```
public String getName(){
  return name;
}

public String getVal(){
  return tstring;
}

public String getValue(){
  return String.valueOf(tstring);
}

public String getPrevString(){
  return String.valueOf(prev_tstring);
}

public String getType(){
  return type;
}

public void setName(String n){
  name = n;
}

public void setType(String t){
  type = t;
}

public void setInit(String init){
   this.init = init;
}
public void setTimeStamp(String ts){
   this.ts = Long.parseLong(ts);
}
public long getTimeStamp(){
   return ts;
}
public void increaseTimeStamp(){
  this.ts ++;
}
public void clearTimeStamp(){
  this.ts = 0;
}
public String getInit(){
    return init;
}
public void setHasBeenInitialized(String hasBeenInitialized){
        this.hasBeenInitialized = hasBeenInitialized;
}
public String getHasBeenInitialized(){
  return            this.hasBeenInitialized;
}
public boolean getChanged(){
  return changed;
}
public void setChanged(boolean flag){
  changed =flag;
```

```
  }
 }
class INTEGER implements SINSVariable{
  private String type = null;
  private String name = null;
  private int prev_yint,yint;
  private String init;
  private String hasBeenInitialized = "false";
  private long ts=0;
  private boolean changed=false;

  public INTEGER(String t,String n,int v){
    setType(t);
    setName(n);
    this.yint = v;
    prev_yint=yint;
  }

  public INTEGER(String t,String n){
    setType(t);
    setName(n);
  }

  public String toString(){
    return String.valueOf(yint);
  }

  public void setValue(int dummy){
    if(type.equals("monitored"))
    {
     if(init.equals("true"))
      {
       prev_yint=dummy;
      yint=dummy;
       hasBeenInitialized = "true";
       }
     else
      {
       prev_yint=yint;
      yint=dummy;
       }
    }
    else
    {
          if(init == null||hasBeenInitialized.equals("false"))
          {
       prev_yint=dummy;
      yint=dummy;
        hasBeenInitialized = "true";
        init = "true";
          }
          else
          {
       prev_yint=yint;
      yint=dummy;
          init = "false";
          }
```

```
  }
  changed = true;
}
public int getPrev(){
  return prev_yint;
}

public void setPrev(int dummy){
  prev_yint=dummy;
}

public String getName(){
  return name;
}

public int getVal(){
  return yint;
}

public String getValue(){
  return String.valueOf(yint);
}

public String getPrevString(){
  return String.valueOf(prev_yint);
}

public String getType(){
  return type;
}

public void setName(String n){
  name = n;
}

public void setType(String t){
  type = t;
}

public void setInit(String init){
   this.init = init;
}
public void setTimeStamp(String ts){
   this.ts = Long.parseLong(ts);
}
public long getTimeStamp(){
   return ts;
}
public void increaseTimeStamp(){
  this.ts ++;
}
public void clearTimeStamp(){
  this.ts = 0;
}
public String getInit(){
    return init;
}
public void setHasBeenInitialized(String hasBeenInitialized){
```

```
            this.hasBeenInitialized = hasBeenInitialized;
    }
    public String getHasBeenInitialized(){
      return            this.hasBeenInitialized;
    }
    public boolean getChanged(){
      return changed;
    }
    public void setChanged(boolean flag){
      changed =flag;
    }
  }
  /* End of Type Definitions */


}
```

TEST RESULTS

The testing process was divided into three stages, the first involved using very simple α-calculus code to ensure the structure of the SOL code generated was correct. The second stage involved writing more complex code in α-calculus, the testing of loops and conditionals was done in this stage. The second stage was the most arduous as the generated SOL code had to be analyzed in detail to ensure that it was syntactically and logically correct. The final testing stage was done with two pieces of α-calculus code that had already been converted to SOL by hand. The hand generated code was compared to the machine generated code to test its correctness. The sample file used was supplied by Arizona State University.

Test Phase 1: syntax and logic testing

Input File 1:

```
fix MA = Name a(Name b).zero
```

Output File 1:

```
module MA{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

Name a;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

internal variables
integer dummyMonitored;

Name b;

definitions

b= initially null  then

if {
[] @C(a) ->
 a

};
}
```

Input File 2:

```
fix MA = integer c<2>.zero
```

Output File 2:

```
module MA{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

integer c;

internal variables
integer dummyMonitored;

definitions

c= initially 0  then if{
[] "true" -> 2

};
}
```

Test Phase 2: syntax, logic, loops and conditionals testing

Input File 3:

```
fix PDAgent = let integer n = integer CAR:dispatchCAR(integer loc, integer where) instantiate let
Name p=Name CAR:setupPerimeter(integer loc) instantiate if p==setup then Name
m<perimeterSetupSuccess>.zero else Name m<perimeterSetupFail>.zero.zero
```

Output File 3:

```
module PDAgent{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

integer n;

Name p;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

Name m;

internal variables
integer dummyMonitored;

definitions

cServiceInvoke = initially null then if {
[]@C( loc) & @C( where) -> "CAR:dispatchCAR(" +  loc + "," +  where + ","+"n" + ")"
[]@C( loc) -> "CAR:setupPerimeter(" +  loc + ","+"p" + ")"

};
m= initially null  then if{

[] @T(p==setup) -> if{
[] "true" ->"perimeterSetupSuccess"}

otherwise ->if{
[] "true" ->"perimeterSetupFail"
}};

}
```

Input File 4:

fix AMBAgent=integer o(integer loc).let Name n=Name AMB:dispatchAMB(integer loc) instantiate Name f(Name people).if people==noInjuredPassenger then Name i<rescued>.zero else let Name status=Name AMB:getInjuryStatus(integer n, Name people) instantiate if status==minor then Name c<notNeedHelicopter>.let Name result=Name AMB:sendHospital(Name people) instantiate Name i<Name result>.zero else Name c<needHelicopter>.Name d<Name people>.zero.zero.zero

Output File 4:

```
module AMBAgent{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

integer o;

Name n;

Name f;

Name status;

Name result;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

Name i;

Name c;

Name d;

internal variables
integer dummyMonitored;

integer loc;

Name people;

definitions

loc= initially 0  then

if {
[] @C(o) ->
 o
};
cServiceInvoke = initially null then if {
[]@C( loc) -> "AMB:dispatchAMB(" +  loc + ","+"n" + ")"

[] "true" ->if{
[]@T(people==noInjuredPassenger) -> otherwise ->if{
```

```
[]@C( n) & @C( people) -> "AMB:getInjuryStatus(" +  n + "," +  people + ","+"status" + ")"
[] "true" ->if{
[] @T(status==minor) -> if{
[]@C( people) -> "AMB:sendHospital(" +  people + ","+"result" + ")"
}
}
}
}
};

people= initially null  then

if {
[] @C(f) ->
 f

};
i= initially null  then if{

[] @T(people==noInjuredPassenger) -> if{
[] "true" ->"rescued"}

otherwise ->if{
[]@T(status==minor) ->
if {
[] @C(result) ->
 result}

}
};

c= initially null  then if{

[] @T(people==noInjuredPassenger)  ->
otherwise -> if{

[] @T(status==minor) -> if{
[] "true" ->"notNeedHelicopter"}

otherwise ->if{
[] "true" ->"needHelicopter"
}}
}
;

d= initially null  then if{
[] @T(people==noInjuredPassenger)  ->
otherwise -> if{

[] @T(status==minor) ->
otherwise -> if{
[] @C(people) ->people
}

}
}
;
}
```

Test Phase 3: testing with files supplied by ASU

Input File 5 – AMBAgent

fix AMBAgent=integer o(integer loc).let Name n=Name AMB:dispatchAMB(integer loc) instantiate Name f(Name people).if people=="then Name i<rescued>.zero else if people>" then let Name status=Name AMB:getInjuryStatus(aldef loc, Name people) instantiate if status=='minor' then Name c<notNeedHelicopter>.let Name result=Name AMB:sendHospital(aldef people) instantiate Name i<Name result>.zero else Name c<needHelicopter>.integer h<aldef loc>.Name d<aldef people>.zero.zero else zero.zero.zero

Output File 5 – AMBAgent.sol

```
module AMBAgent{
type definitions
tString=string;
Name=string;
monitored variables
integer dummyMonitored;

integer o;

Name n;

Name f;

Name status;

Name result;

controlled variables
integer dummyMonitored;
tString cServiceInvoke;

Name i;

Name c;

integer h;

Name d;

internal variables
integer dummyMonitored;

integer loc;

Name people;

definitions

loc= initially 0  then

if {
[] @C(o) ->
 o
```

```
};
cServiceInvoke = initially null then if {
[]@C( loc) -> "AMB:dispatchAMB(" +  loc + ","+"n" + ")"

[] "true" ->if{
[]@T(people=="") -> otherwise ->if{
[] "true" ->if{
[] @T(people>"") -> if{
[]@C( people) -> "AMB:getInjuryStatus(" +  loc + "," +  people + ","+"status" + ")"
[] "true" ->if{
[] @T(status=="minor") -> if{
[] -> "AMB:sendHospital(" +  people + ","+"result" + ")"
}
}
}
}
}
}
};

people= initially null  then

if {
[] @C(f) ->
 f

};
i= initially null  then if{


[] @T(people=="") -> if{
[] "true" ->"rescued"}

otherwise ->if{
[]@T(people>"") -> if{
[]@T(status=="minor") ->
if {
[] @C(result) ->
 result}

}


}
};


c= initially null  then if{

[] @T(people=="")  ->
otherwise -> if{
[] @T(people>"")  -> if{


[] @T(status=="minor") -> if{
[] "true" ->"notNeedHelicopter"}

otherwise ->if{
```

```
[] "true" ->"needHelicopter"
}}
}

}
;


h= initially 0  then if{
[] @T(people=="")  ->
otherwise -> if{
[] @T(people>"")  -> if{


[] @T(status=="minor") ->
otherwise -> if{

[] "true" ->loc}

}
}

}
;
d= initially null  then if{
[] @T(people=="")  ->
otherwise -> if{
[] @T(people>"")  -> if{


[] @T(status=="minor") ->
otherwise -> if{

[] "true" ->people}

}
}

}
;
}
```

CONCLUSION

The concept of service based systems and web services is fairly new, whether it will be the future of the computer industry or the flavor of the week remains to be seen. In the meantime, is the approach described in my thesis the proverbial silver bullet solution, which software engineers keep talking about, to all our service based system needs? In a short answer yes, it certainly allows a developer to create provable code in a low level language that can be ported into a high level language and used in multiple scenarios.

But, the technique that makes this solution work is the same one that could cause problems in some cases. α-calculus is not a very common programming language like C or C++; hence a developer would need to learn how to code in α-calculus first. The language is still fairly easy to learn, although some of the syntax can be counterintuitive at times. Naturally one would ask why not just build the translator from C/C++, the reason is that those and other high level languages are not provable and as a result it would serve no purpose to develop a translator from those languages to SOL. In a sense the learning curve to master α-calculus is a very small price to pay to get provable code. Finally, even though α-calculus is provable, the translator from it to SOL hasn't been proven yet. Naturally, given more time this would be the next step in forming an even more robust solution.

Work has been completed on a SOL to Java translator and a SOL to BPEL translator is being developed. This opens up further expansion possibilities for the α-calculus to SOL translator. One can now deploy Java or BPEL source code for business processes that are provable.

Finally, was using Gentle as a construction tool the best idea? Yes, when I checked the C source code generated by Gentle I quickly realized that coding this in a more standard language like C or C++ would have taken infinitely longer. Using a language like Gentle helped reduce the number of lines of code, and as a result made the debugging process a whole lot easier.

BIBLIOGRAPHY

1. Friedrich Wilhelm Schröer. *The GENTLE Compiler Construction System.* R. Oldenbourg Verlag, Munich and Vienna, 1997. ISBN 3-486-24703-4

2. *The GENTLE Compiler Construction Website.* http://gentle.compilertools.net

3. Ramesh Bharadwaj. *SOL: A Verifiable Synchronous Language for Reactive Systems.*

4. Ramesh Bharadwaj. *SINS: A Middleware for Autonomous Agents and Secure Code Mobility.*

5. Martin Deubler, Johannes Grunbauer, Jan Jurjens and Guido Wimmel. *Sound Development of Secure Service Based Systems.* Proceedings of the 2nd international conference on Service oriented computing, New York, 2004. ISBN 1-58113-871-7

6. Ramesh Bharadwaj and Supratik Mukhopadhyay. *Functional "AJAX" in Secure Synchronous Programming.* WWW2006, Edinburgh, UK, 2006.

7. Donald I. Good. *Provable Programming.* Proceedings of the international conference on reliable software, Los Angeles, 1975.

8. Thomas J. Bergin, Richard G. Gibson, Richard G. Gibson Jr. *History of Programming Languages.* Addison-Wesley Professional, 1996. ISBN 0201895021.

9. Supratik Mukhopadhyay. *A Calculus and a Logic For Service Based Systems*

10. *Business Process Execution Language for Web Services Website.* http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

11. Ramesh Bharadwaj and Supratik Mukhopadhyay. *Position Paper: Formal Methods for Developing Adaptable, Secure, Situation-aware Service-oriented (AS^3) Architectures.*

12. *The SALSA Website.* http://www.reactive-systems.com/salsa

SOURCE CODE

```
'root' Program(->P)
print(P)
stack <- 0
newstack <- 0
OpenOutput("out.sol")
Pretranslate(P)
CloseOutput

'type' IDENT

'type' TYPE
type(IDENT)
func(TYPE,TYPE)

'type' CHANNEL
channel(VAR)

'type' SYSTEM
par(SYSTEM,SYSTEM)
atsys1(IDENT,PROCESS)
fail1(IDENT,PROCESS)
nil

'type' PROCESS
zero
parproc(PROCESS,PROCESS)
dotext(EXT_ACT,PROCESS)
dotint(INT_COMP,PROCESS)
plus(PROCESS,PROCESS)
dotcat(CATCH,PROCESS)
failure(IDENT)
dottime(TIMEOUT,PROCESS)
id_process(IDENT)

'type' EXT_ACT
move(M)
comm(K)

'type' GEN_ACT
genexact(EXT_ACT)
genincomp(INT_COMP)
dotgenact(GEN_ACT,GEN_ACT)

'type' INT_COMP
let_instantiate(VAR,BASIC_INT_COMP,GEN_ACT)
if_then_else(BOOLEAN_INT_COMP,PROCESS,PROCESS)
assign(TYPE,IDENT,IDENT,TYPE,IDENT,IDENT)
epsilon
bottom

'type' BASIC_INT_COMP
method(TYPE,IDENT,IDENT,VARLIST)

'type' BOOLEAN_INT_COMP
cons(CONSTRAINT)
tt
ff

'type' CONSTRAINT
gtc(UNTYPED_VAR,INT)
ltc(UNTYPED_VAR,INT)
geq(UNTYPED_VAR,INT)
leq(UNTYPED_VAR,INT)
eqeqcons(UNTYPED_VAR,INT)
gtcv(UNTYPED_VAR,UNTYPED_VAR)
ltcv(UNTYPED_VAR,UNTYPED_VAR)
geqv(UNTYPED_VAR,UNTYPED_VAR)
leqv(UNTYPED_VAR,UNTYPED_VAR)
eqeq(UNTYPED_VAR,UNTYPED_VAR)
gtcvc(UNTYPED_VAR,UNTYPED_VAR,INT)
ltcvc(UNTYPED_VAR,UNTYPED_VAR,INT)
```

```
geqvc(UNTYPED_VAR,UNTYPED_VAR,INT)
leqvc(UNTYPED_VAR,UNTYPED_VAR,INT)
eqeqc(UNTYPED_VAR,UNTYPED_VAR,INT)
and(CONSTRAINT,CONSTRAINT)

'type' M
in(N)
out(N)
open(N)
epsilon

'type' K
bracket(CHANNEL,VAR)
angle(CHANNEL,VAR)
send(CHANNEL,IDENT)
send1(CHANNEL,INT)

'type' N
name_variable(IDENT)
name(IDENT)

'type' VARLIST
varlist(VAR,VARLIST)
nil

'type' UNTYPED_VAR
untyped_var(IDENT)
untyped_string(IDENT)
empty_string

'type' VAR
var(TYPE,IDENT)

'type' CATCH
catch(IDENT)

'type' TIMEOUT
timeout(INT)

'type' ID
id(IDENT)

'type' VARPAIR
varpair(CHANNEL,INT)

'type' BOOLEAN_INT_COMP_PAIR
intpair(BOOLEAN_INT_COMP,INT)

'type' CHECKLIST
lst(VARPAIR,CHECKLIST)
nil

'type' BOOLEAN_INT_COMP_LIST
conlist(BOOLEAN_INT_COMP_LIST,BOOLEAN_INT_COMP_PAIR)
nil
'type' CHANLIST
chanlist(CHANNEL,CHANLIST)
nil

'var' checkl:CHECKLIST
'var' ifc: BOOLEAN_INT_COMP_LIST
'var' ifc1: BOOLEAN_INT_COMP_LIST

'token' Ident(->IDENT)
'token' Number(->INT)

'var' stack: INT
'var' newstack: INT
'var' newstackvar: INT
'var' newstackserv: INT
'var' service: INT
'var' TempConsList: BOOLEAN_INT_COMP_LIST
'var' VarConsList: BOOLEAN_INT_COMP_LIST
```

```
'var' ServConsList: BOOLEAN_INT_COMP_LIST
'var' declaredvariable: CHANLIST
'var' countif:INT
'var' countifinner:INT

/*--------------------NON TERMINALS-----------------------------------------------*/
/*---------------------PROGRAM-----------------------------------------------------*/

'nonterm' Program(->SYSTEM)
        'rule' Program(->P):
                                                                System(->P)


/*---------------------SYSTEM------------------------------------------------------------*/

'nonterm' System(->SYSTEM)
                'rule' System(->par(S1,S2)):
                                        System(->S1)
                                        "||"
                                        System(->S2)
                'rule' System(->atsys1(I,P)):
                                        "fix"
                                        Ident(->I)
                                        "="
                                        Process(->P)
                'rule' System(->fail1(I,P)):
                                        "fix fail"
                                        Ident(->I)
                                        "="
                                        Process(->P)


/*--------------------PROCESS-----------------------------------------------------------*/

'nonterm' Process(->PROCESS)
                'rule' Process(->parproc(P,Q)):
                                        Process(->P)
                                        "par"
                                        Process(->Q)
                'rule' Process(->dotext(E,P)):
                                        ExtAct(->E)
                                        "."
                                        Process(->P)
                'rule' Process(->dotint(C,P)):
                                        IntComp(->C)
                                        "."
                                        Process(->P)
                'rule' Process(->plus(P,Q)):
                                        Process(->P)
                                        "pl"
                                        Process(->Q)
                'rule' Process(->dotcat(catch(I),P)):
                                        "catch"
                                        "("
                                        Ident(->I)
                                        ")"
                                        "."
                                        Process(->P)
                'rule' Process(->failure(I)):
                                        "failref"
                                        "("
                                        Ident(->I)
                                        ")"
                'rule' Process(->dottime(T,P)):
                                        "timeout"
                                        Time1(->T)
                                        "."
                                        Process(->P)
                'rule' Process(->id_process(P)):
                                        Ident(->P)
                'rule' Process(->zero):
                                        "zero"
```

```
/*-------------------------------------EXTERNAL ACTION-----------------------------------------------------*/

'nonterm' ExtAct(->EXT_ACT)
        'rule' ExtAct(->move(in(name_variable(N)))):
                                "in"
                                "name"
                                Ident(->N)
        'rule' ExtAct(->move(in(name(N)))):
                                "in"
                                Ident(->N)
        'rule' ExtAct(->move(out(name_variable(N)))):
                                "out"
                                "name"
                                Ident(->N)
        'rule' ExtAct(->move(out(name(N)))):
                                "out"
                                Ident(->N)
        'rule' ExtAct(->move(open(name_variable(N)))):
                                "open"
                                "name"
                                Ident(->N)
        'rule' ExtAct(->move(open(name(N)))):
                                "open"
                                Ident(->N)
        'rule' ExtAct(->comm(bracket(C,V))):
                                Channel(->C)
                                "("
                                Var(->V)
                                ")"
        'rule' ExtAct(->comm(angle(C,V))):
                                Channel(->C)
                                "<"
                                Var(->V)
                                ">"
        'rule' ExtAct(->comm(send(C,U))):
                                Channel(->C)
                                "<"
                                Ident(->U)
                                ">"
        'rule' ExtAct(->comm(send1(C,U))):
                                Channel(->C)
                                "<"
                                Number(->U)
                                ">"


/*-------------------------------------INTERNAL COMPUTATION------------------------------------------------*/

'nonterm' IntComp(->INT_COMP)
        'rule' IntComp(->let_instantiate(var(T,X),method(T1,I,J,varlist(V,L)),C)):
                                "let"
                                Type(->T)
                                Ident(->X)
                                "="
                                Type(->T1)
                                Ident(->I)
                                ":"
                                Ident(->J)
                                "("
                                Var(->V)
                                ","
                                Varlist(->L)
                                "instantiate"
                                GenAct(->C)
                                eq(T,T1)
        'rule' IntComp(->let_instantiate(var(T,X),method(T1,I,J,varlist(V,nil)),C)):
                                "let"
                                Type(->T)
                                Ident(->X)
                                "="
                                Type(->T1)
                                Ident(->I)
```

```
                                        ":"
                                        Ident(->J)
                                        "("
                                        Var(->V)
                                        ")"
                                        "instantiate"
                                        GenAct(->C)
                                        eq(T,T1)
                'rule' IntComp(->let_instantiate(var(T,X),method(T1,I,J,nil),C)):
                                        "let"
                                        Type(->T)
                                        Ident(->X)
                                        "="
                                        Type(->T1)
                                        Ident(->I)
                                        ":"
                                        Ident(->J)
                                        "("
                                        ")"
                                        "instantiate"
                                        GenAct(->C)
                                        eq(T,T1)
                'rule' IntComp(->if_then_else(cons(Cons),P,Q)):
                                        "if"
                                        Constraint(->Cons)
                                        "then"
                                        Process(->P)
                                        "else"
                                        Process(->Q)
                'rule' IntComp(->assign(T1,I,J,T2,K,L)):
                                        Type(->T1)
                                        Ident(->I)
                                        ":"
                                        Ident(->J)
                                        "<-"
                                        Type(->T2)
                                        Ident(->K)
                                        ":"
                                        Ident(->L)
                                        eq(T1,T2)


/*-----------------------------------GENERAL ACTIONS-------------------------------------------------------------*/

'nonterm' GenAct(->GEN_ACT)
                'rule' GenAct(->genexact(E)):
                                        ExtAct(->E)
                'rule' GenAct(->genincomp(C)):
                                        IntComp(->C)
                'rule' GenAct(->dotgenact(C,D)):
                                        GenAct(->C)
                                        "dot"
                                        GenAct(->D)



/*-----------------------------------CHANNEL---------------------------------------------------------------------*/

'nonterm' Channel(->CHANNEL)
                'rule' Channel(->channel(C)):
                                        Var(->C)


/*-----------------------------------VARIABLE LIST---------------------------------------------------------------*/

'nonterm' Varlist(->VARLIST)
                'rule' Varlist(->varlist(V,Vs)):
                                        Var(->V)
                                        ","
                                        Varlist(->Vs)
                'rule' Varlist(->varlist(V,nil)):
                                        Var(->V)
                                        ")"
```

```
/*--------------------------------CONSTRAINTS-----------------------------------------------------------------------*/

'nonterm' Constraint(->CONSTRAINT)
        'rule' Constraint(->gtc(U,V)):
                                Untyped_Var(->U)
                                ">"
                                Number(->V)
        'rule' Constraint(->ltc(U,V)):
                                Untyped_Var(->U)
                                "<"
                                Number(->V)
        'rule' Constraint(->geq(U,V)):
                                Untyped_Var(->U)
                                ">="
                                Number(->V)
        'rule' Constraint(->leq(U,V)):
                                Untyped_Var(->U)
                                "=<"
                                Number(->V)
        'rule' Constraint(->eqeqcons(U,V)):
                                Untyped_Var(->U)
                                "=="
                                Number(->V)
        'rule' Constraint(->gtcv(U,V)):
                                Untyped_Var(->U)
                                ">"
                                Untyped_Var(->V)
        'rule' Constraint(->ltcv(U,V)):
                                Untyped_Var(->U)
                                "<"
                                Untyped_Var(->V)
        'rule' Constraint(->leqv(U,V)):
                                Untyped_Var(->U)
                                "=<"
                                Untyped_Var(->V)
        'rule' Constraint(->geqv(U,V)):
                                Untyped_Var(->U)
                                ">="
                                Untyped_Var(->V)
        'rule' Constraint(->eqeq(U,V)):
                                Untyped_Var(->U)
                                "=="
                                Untyped_Var(->V)
        'rule' Constraint(->gtcvc(U,V,C)):
                                Untyped_Var(->U)
                                ">"
                                Untyped_Var(->V)
                                "+"
                                Number(->C)
        'rule' Constraint(->ltcvc(U,V,C)):
                                Untyped_Var(->U)
                                "<"
                                Untyped_Var(->V)
                                "+"
                                Number(->C)
        'rule' Constraint(->leqvc(U,V,C)):
                                Untyped_Var(->U)
                                "=<"
                                Untyped_Var(->V)
                                "+"
                                Number(->C)
        'rule' Constraint(->geqvc(U,V,C)):
                                Untyped_Var(->U)
                                ">="
                                Untyped_Var(->V)
                                "+"
                                Number(->C)
        'rule' Constraint(->eqeqc(U,V,C)):
                                Untyped_Var(->U)
                                "=="
                                Untyped_Var(->V)
```

```
                                        "+"
                                        Number(->C)
                'rule' Constraint(->and(U,V)):
                                        Constraint(->U)
                                        "&&"
                                        Constraint(->V)


/*-----------------------------------UNTYPED VARIABLE-----------------------------------------------------*/

'nonterm' Untyped_Var(->UNTYPED_VAR)
                'rule' Untyped_Var(->untyped_string(X)):
                                        """
                                        Ident(->X)
                                        """
                'rule' Untyped_Var(->empty_string):
                                        """
                                        """
                'rule' Untyped_Var(->untyped_var(X)):
                                        Ident(->X)



/*-----------------------------------------VARIABLE-------------------------------------------------------*/

'nonterm' Var(->VAR)
                'rule'   Var(->var(T,X)):
                                        Type(->T)
                                        Ident(->X)


/*-------------------------------------------------TYPE-----------------------------------------------------*/

'nonterm' Type(->TYPE)
                'rule' Type(->type(T)):
                                        Ident(->T)
                                        id_to_string(T->T1)
                                        (| eq(T1,"integer")
                                        || eq(T1,"real")
                                        || eq(T1,"Name")
                                        || eq(T1,"bool")
                                        || eq(T1,"aldef")|)
                'rule' Type(->func(T,S)):
                                        Type(->T)
                                        "->"
                                        Type(->S)



/*-----------------------------------------TIME OUT-------------------------------------------------------*/

'nonterm' Time1(->TIMEOUT)
                'rule' Time1(->timeout(T)):
                                        Number(->T)

'var' Temp: STRING

'action' OpenOutput(STRING)
'action' CloseOutput
'action' Put(STRING)
'action' PutI(INT)
'action' Nl
'action' Kprintc(STRING)
'action' Kprints(IDENT)


/*-----------------------------------TRANSLATION----------------------------------------------------------*/

'action' Pretranslate(Pr:SYSTEM)
                'rule' Pretranslate(Pr):
                                          Translate(Pr)
```

```
'action' Translate(Pr: SYSTEM)
        'rule' Translate(par(S1,S2)):
                                Translate(S1)
                                Translate(S2)
        'rule' Translate(atsys1(I,P)):
                                Nl service<-0
                                checkl<-nil
                                ifc<-nil
                                ifc1<-nil
                                TempConsList<-nil
                                VarConsList<- nil
                                ServConsList<-nil
                                newstack<-0
                                newstackvar<-0
                                newstackserv<-0
                                countif<-0
                                countifinner<-0
                                declaredvariable<- nil
                                Put("module ")
                                id_to_string(I->M)
                                Put(M)Put("{")
                                Nl Put("type definitions")
                                Nl Put("tString=string;")
                                Nl Put("Name=string;")
                                Nl Put("monitored variables")
                                Nl Put("integer dummyMonitored;")
                                Nl CollectMonitoredVariable(P)
                                Nl Put("controlled variables")
                                Nl Put("integer dummyMonitored;")
                                Nl Put("tString cServiceInvoke;")
                                Nl CollectControlledVariable(P)
                                Nl Put("internal variables")
                                Nl Put("integer dummyMonitored;")
                                Nl CollectInternalVariable(P)
                                Nl Put("definitions")
                                Nl TranslateProcess(P)
                                Nl Put("}")
                                Nl service<-0
        'rule' Translate(fail1(I,P)):
                                Nl service<-0
                                checkl<-nil
                                ifc<-nil
                                TempConsList<-nil
                                VarConsList<- nil
                                ServConsList<-nil
                                newstack<-0
                                newstackvar<-0
                                newstackserv<-0
                                declaredvariable<- nil
                                countif<-0
                                Put("module ")
                                id_to_string(I->M)
                                Put("fail_")
                                Put(M)
                                Put("{")
                                Nl Put("type definitions")
                                Nl Put("tString=string;")
                                Nl Put("monitored variables")
                                Nl CollectMonitoredVariable(P)
                                Nl Put("controlled variables")
                                Nl Put("tString cServiceInvoke;")
                                Nl CollectControlledVariable(P)
                                Nl Put("internal variables")
                                Nl CollectInternalVariable(P)
                                Nl Put("definitions")
                                Nl TranslateProcess(P)
                                Nl Put("}")
                                Nl service<- 0


/*-----------------------------------TRANSLATE A PROCESS-----------------------------------------------------------*/
```

```
'action' TranslateProcess(P: PROCESS)
        'rule' TranslateProcess(zero)
        'rule' TranslateProcess(dotext(E,P)):
                                TranslateExt(E,P)
                                TranslateProcess(P)
                                (| IsStack(0)
                                Put("")
                                || Put("}")
                                stack->N
                                stack<-N-1 |)
        'rule' TranslateProcess(dotint(C,P)):
                                TranslateInt(C,P)
                                TranslateProcess(P)
        'rule' TranslateProcess(parproc(P,Q)):
                                TranslateProcess(P)
                                TranslateProcess(Q)


/*----------------------------------COLLECT MONITORED VARIABLES----------------------------------------------------*/

'action' CollectMonitoredVariable(P:PROCESS)
        'rule' CollectMonitoredVariable(zero)
        'rule' CollectMonitoredVariable(dotext(E,P)):
                                CollectMonitoredExt(E)
                                CollectMonitoredVariable(P)
        'rule' CollectMonitoredVariable(dotint(C,P)):
                                CollectMonitoredInt(C)
                                CollectMonitoredVariable(P)
        'rule' CollectMonitoredVariable(parproc(P,Q)):
                                CollectMonitoredVariable(P)
                                CollectMonitoredVariable(Q)


/*----------------------------------COLLECT MONITORED VARIABLES EXTERNAL ACTIONS----------------------------------*/

'action' CollectMonitoredExt(E: EXT_ACT)
        'rule' CollectMonitoredExt(comm(bracket(C,V))):
                                (|checkdeclaredvariable(C)
                                PrintChannel(C)
                                declaredvariable -> L
                                declaredvariable <- chanlist(C,L)
                                || |)
        'rule' CollectMonitoredExt(comm(angle(C,V)))
        'rule' CollectMonitoredExt(comm(send(C,U)))
        'rule' CollectMonitoredExt(comm(send1(C,U)))


/*----------------------------------COLLECT MONITORED VARIABLES INTERNAL COMPUTATIONS------------------------------*/

'action' CollectMonitoredInt(C:INT_COMP)
        'rule' CollectMonitoredInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),C)):
                                Nl id_to_string(T->T0)
                                Put(T0)
                                id_to_string(X->K)
                                Put(" ")
                                Put(K)
                                Put(";")
                                Nl CollectMonitoredGen(C)
        'rule' CollectMonitoredInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),C)):
                                Nl id_to_string(T->T0)
                                Put(T0)
                                id_to_string(X->K)
                                Put(" ")
                                Put(K)
                                Put(";")
                                Nl CollectMonitoredGen(C)
        'rule' CollectMonitoredInt(if_then_else(cons(Cons),P,Q)):
                                CollectMonitoredVariable(P)
                                CollectMonitoredVariable(Q)
```

```
/*----------------------------------COLLECT MONITORED VARIABLES GENERAL ACTIONS----------------------------------*/

'action' CollectMonitoredGen(C:GEN_ACT)
        'rule' CollectMonitoredGen(genexact(C)):
                                CollectMonitoredExt(C)
        'rule' CollectMonitoredGen(genincomp(C)):
                                CollectMonitoredInt(C)
        'rule' CollectMonitoredGen(dotgenact(C,D)):
                                CollectMonitoredGen(C)
                                CollectMonitoredGen(D)


/*----------------------------------PRINT VARIABLES----------------------------------------------------------------*/

'action' PrintVar(C: VAR)
        'rule' PrintVar(var(type(T),C)):
                                Nl id_to_string(T->T0)
                                Put(T0)
                                Put(" ")
                                id_to_string(C->D)
                                Put(D)
                                Put(";") Nl


/*----------------------------------COLLECT INTERNAL VARIABLES-----------------------------------------------------*/

'action' CollectInternalVariable(P: PROCESS)
        'rule' CollectInternalVariable(zero)
        'rule' CollectInternalVariable(dotext(E,P)):
                                CollectInternalExt(E)
                                CollectInternalVariable(P)
        'rule' CollectInternalVariable(dotint(C,P)):
                                CollectInternalInt(C)
                                CollectInternalVariable(P)
        'rule' CollectInternalVariable(parproc(P,Q)):
                                CollectInternalVariable(P)
                                CollectInternalVariable(Q)


/*----------------------------------COLLECT INTERNAL VARIABLES EXTERNAL ACTIONS-------------------------------------*/

'action' CollectInternalExt(E: EXT_ACT)
        'rule'  CollectInternalExt(comm(bracket(C,V))):
                                PrintVar(V)
        'rule' CollectInternalExt(comm(angle(C,V)))
        'rule' CollectInternalExt(comm(send(C,U)))
        'rule' CollectInternalExt(comm(send1(C,U)))


/*----------------------------------COLLECT INTERNAL VARIABLES INTERNAL COMPUTATIONS--------------------------------*/

'action' CollectInternalInt(C:INT_COMP)
        'rule' CollectInternalInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),C)):
                                CollectInternalGen(C)
        'rule' CollectInternalInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),C)):
                                CollectInternalGen(C)
        'rule' CollectInternalInt(if_then_else(cons(Cons),P,Q)):
                                CollectInternalVariable(P)
                                CollectInternalVariable(Q)


/*----------------------------------COLLECT INTERNAL VARIABLES GENERAL ACTIONS-------------------------------------*/

'action' CollectInternalGen(C:GEN_ACT)
        'rule' CollectInternalGen(genexact(C)):
                                CollectInternalExt(C)
        'rule' CollectInternalGen(genincomp(C)):
```

```
                                CollectInternalInt(C)
        'rule' CollectInternalGen(dotgenact(C,D)):
                                CollectInternalGen(C)
                                CollectInternalGen(D)


/*--------------------------------COLLECT CONTROLLED VARIABLES-----------------------------------------------------------*/

'action' CollectControlledVariable(P: PROCESS)
        'rule' CollectControlledVariable(zero)
        'rule' CollectControlledVariable(dotext(E,P)):
                                CollectControlledExt(E)
                                CollectControlledVariable(P)
        'rule' CollectControlledVariable(dotint(C,P)):
                                CollectControlledInt(C)
                                CollectControlledVariable(P)
        'rule' CollectControlledVariable(parproc(P,Q)):
                                CollectControlledVariable(P)
                                CollectControlledVariable(Q)


/*--------------------------------COLLECT CONTROLLED VARIABLES EXTERNAL ACTIONS-------------------------------------*/

'action' CollectControlledExt(E: EXT_ACT)
        'rule'  CollectControlledExt(comm(bracket(C,V)))
        'rule' CollectControlledExt(comm(angle(C,V))):
                                (|checkdeclaredvariable(C)
                                PrintChannel(C)
                                declaredvariable -> L
                                declaredvariable <- chanlist(C,L)
                                || |)
        'rule' CollectControlledExt(comm(send(C,U))):
                                (|checkdeclaredvariable(C)
                                PrintChannel(C)
                                declaredvariable -> L
                                declaredvariable <- chanlist(C,L)
                                || |)
        'rule' CollectControlledExt(comm(send1(C,U))):
                                (|checkdeclaredvariable(C)
                                PrintChannel(C)
                                declaredvariable -> L
                                declaredvariable <- chanlist(C,L)
                                || |)


/*--------------------------------COLLECT CONTROLLED VARIABLE INTERNAL COMPUTATIONS---------------------------------*/

'action' CollectControlledInt(C:INT_COMP)
        'rule' CollectControlledInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),C)):
                                CollectControlledGen(C)
        'rule' CollectControlledInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),C)):
                                CollectControlledGen(C)
        'rule' CollectControlledInt(if_then_else(cons(Cons),P,Q)):
                                CollectControlledVariable(P)
                                CollectControlledVariable(Q)


/*--------------------------------COLLECT CONTROLLED VARIABLE GENERAL ACTIONS---------------------------------------*/

'action' CollectControlledGen(C:GEN_ACT)
        'rule' CollectControlledGen(genexact(C)):
                                CollectControlledExt(C)
        'rule' CollectControlledGen(genincomp(C)):
                                CollectControlledInt(C)
        'rule' CollectControlledGen(dotgenact(C,D)):
                                CollectControlledGen(C)
                                CollectControlledGen(D)


/*--------------------------------TRANSLATE EXTERNAL ACTION--------------------------------------------------------*/
```

```
'action' TranslateExt(E: EXT_ACT, P:PROCESS)
        'rule' TranslateExt(comm(bracket(C,V)),P):
                        Nl InsertVar(V)
                        InsertatC(C)
                        PrintVarEq(V,C)
                        Nl Put("};")
        'rule' TranslateExt(comm(angle(C,V)),P):
                        (|ChecknotVar(C)
                        checkl->L
                        checkl<-lst(varpair(C,1),L)
                        PrintVarChanEq(C,V)
                        PrintAllDefinitionsfromRest(C,P)
                        Nl Put("};")
                        || |)
        'rule' TranslateExt(comm(send(C,U)),P):
                        (|ChecknotVar(C)
                        checkl->L
                        checkl<- lst(varpair(C,1),L)
                        PrintChanConstantEq(C,U)
                        PrintAllDefinitionsfromRest(C,P)
                        Nl Put("};")
                        ||  |)
        'rule' TranslateExt(comm(send1(C,U)),P):
                        (|ChecknotVar(C)
                        checkl->L
                        checkl<- lst(varpair(C,1),L)
                        PrintChanIntEq(C,U)
                        PrintAllDefinitionsfromRest(C,P)
                        Nl Put("};")
                        ||  |)


/*-----------------------------PRINT ALL DEFINITIONS FROM REST-----------------------------------------------------------*/

'action' PrintAllDefinitionsfromRest(C: CHANNEL,P:PROCESS)
        'rule'  PrintAllDefinitionsfromRest(channel(var(T,X)),zero)
        'rule'  PrintAllDefinitionsfromRest(channel(var(T,X)), dotext(E,P)):
                        PrintAllDefinitionsfromRestExt(var(T,X),E,P)
                        PrintAllDefinitionsfromRest(channel(var(T,X)),P)
        'rule'  PrintAllDefinitionsfromRest(channel(var(T,X)), dotint(C1,P)):
                        PrintAllDefinitionsfromRestInt(var(T,X),C1,P)
        'rule'  PrintAllDefinitionsfromRest(channel(var(T,X)), parproc(P,Q))


/*-----------------------------PRINT ALL DEFINITIONS FROM REST EXTERNAL ACTIONS-------------------------------------------*/

'action' PrintAllDefinitionsfromRestExt(C:VAR, E:EXT_ACT,P:PROCESS)
        'rule'  PrintAllDefinitionsfromRestExt(var(T,X),comm(bracket(C,V)),P)
        'rule'  PrintAllDefinitionsfromRestExt(var(T1,X1),comm(angle(channel(var(T,X)),V)),P):
                        (|eq(T,T1)
                        eq(X,X1)
                        InsertatCV(V)
                        PrintvariableNoType(V)
                        Put("}")
                        || |)
        'rule'  PrintAllDefinitionsfromRestExt(var(T,X),comm(send(channel(var(T1,X1)),U)),P):
                        (|eq(T,T1)
                        eq(X,X1)
                        Insertattrue(channel(var(T1,X1)))
                        id_to_string(U->U1)
                        Put("\"")
                        Put(U1)
                        Put("\"")
                        Nl Put("}")
                        || |)
        'rule'  PrintAllDefinitionsfromRestExt(var(T,X),comm(send1(channel(var(T1,X1)),U)),P):
                        (|eq(T,T1)
                        eq(X,X1)
                        Insertattrue(channel(var(T1,X1)))
                        PutI(U)
```

```
                                    Nl Put("}")
                                    || |)


/*----------------------------PRINT ALL DEFINITIONS FROM REST EXT IF-----------------------------------------------------*/

'action' PrintAllDefinitionsfromRestExtIf(C:VAR, E:EXT_ACT,P:PROCESS)
            'rule'   PrintAllDefinitionsfromRestExtIf(var(T,X),comm(bracket(C,V)),P)
            'rule'   PrintAllDefinitionsfromRestExtIf(var(T1,X1),comm(angle(channel(var(T,X)),V)),P):
                                    (leq(T,T1)
                                    eq(X,X1)
                                    VarConsList->L
                                    ne(L,nil)
                                    countifinner<-1
                                    countif<-1
                                    InsertatCV(V)
                                    PrintvariableNoType(V)
                                    Put("}")
                                    || eq(T,T1)
                                    eq(X,X1)
                                    countifinner<-1
                                    countif<-1
                                    InsertatCV(V)
                                    PrintvariableNoType(V)
                                    Put("}")
                                    || |)
            'rule'   PrintAllDefinitionsfromRestExtIf(var(T,X),comm(send(channel(var(T1,X1)),U)),P):
                                    (leq(T,T1)
                                    eq(X,X1)
                                    VarConsList->L
                                    ne(L,nil)
                                    countifinner<-1
                                    countif<-1
                                    Insertattrue(channel(var(T1,X1)))
                                    id_to_string(U->U1)
                                    Put("\"")
                                    Put(U1)
                                    Put("\"")
                                    Nl Put("}")
                                    || eq(T,T1)
                                    eq(X,X1)
                                    countifinner<-1
                                    countif<-1
                                    Insertattrue(channel(var(T1,X1)))
                                    id_to_string(U->U1)
                                    Put(U1)
                                    Nl Put("}")
                                    || |)
            'rule'   PrintAllDefinitionsfromRestExtIf(var(T,X),comm(send1(channel(var(T1,X1)),U)),P):
                                    (leq(T,T1)
                                    eq(X,X1)
                                    VarConsList->L
                                    ne(L,nil)
                                    countifinner<-1
                                    countif<-1
                                    Insertattrue(channel(var(T1,X1)))
                                    PutI(U)
                                    Nl Put("}")
                                    || eq(T,T1)
                                    eq(X,X1)
                                    countifinner<-1
                                    countif<-1
                                    Insertattrue(channel(var(T1,X1)))
                                    PutI(U)
                                    Nl Put("}")
                                    || |)


/*----------------------------------PRINT ALL DEFINITIONS FROM REST EXT OTHERWISE---------------------------------*/

'action' PrintAllDefinitionsfromRestExtOtherwise(C:VAR, E:EXT_ACT,P:PROCESS)
```

```
        'rule'   PrintAllDefinitionsfromRestExtOtherwise(var(T,X),comm(bracket(C,V)),P)
        'rule'   PrintAllDefinitionsfromRestExtOtherwise(var(T1,X1),comm(angle(channel(var(T,X)),V)),P):
                                (|eq(T,T1)
                                eq(X,X1)
                                InsertatCV(V)
                                PrintvariableNoType(V)
                                Put("}")
                                || |)
        'rule'   PrintAllDefinitionsfromRestExtOtherwise(var(T,X),comm(send(channel(var(T1,X1)),U)),P):
                                (|eq(T,T1)
                                eq(X,X1)
                                Insertattrue(channel(var(T1,X1)))
                                id_to_string(U->U1)
                                Put("\"")
                                Put(U1)
                                Put("\"")
                                Nl Put("}")
                                || |)
        'rule'   PrintAllDefinitionsfromRestExtOtherwise(var(T,X),comm(send1(channel(var(T1,X1)),U)),P):
                                (|eq(T,T1)
                                eq(X,X1)
                                Insertattrue(channel(var(T1,X1)))
                                PutI(U)
                                Nl Put("}")
                                || |)


/*----------------------------PRINT ALL DEFINITIONS FROM REST INTERNAL COMPUTATIONS-------------------------------------*/

'action' PrintAllDefinitionsfromRestInt(C:VAR, I:INT_COMP,P:PROCESS)
        'rule' PrintAllDefinitionsfromRestInt(var(T,X),
let_instantiate(var(type(T1),X1),method(T2,I,J,varlist(V,L)),genincomp(C)),P):
                                PrintAllDefinitionsfromRestInt(var(T,X),C,P)
                                PrintAllDefinitionsfromRest(channel(var(T,X)),P)
        'rule' PrintAllDefinitionsfromRestInt(var(T,X),
let_instantiate(var(type(T1),X1),method(T2,I,J,varlist(V,L)),genexact(C)),P):
                                PrintAllDefinitionsfromRestExt(var(T,X),C,P)
                                PrintAllDefinitionsfromRest(channel(var(T,X)),P)
        'rule' PrintAllDefinitionsfromRestInt(var(T,X), let_instantiate(var(type(T1),X1),method(T2,I,J,nil),genincomp(C)),P):
                                PrintAllDefinitionsfromRestInt(var(T,X),C,P)
                                PrintAllDefinitionsfromRest(channel(var(T,X)),P)
        'rule' PrintAllDefinitionsfromRestInt(var(T,X), let_instantiate(var(type(T1),X1),method(T2,I,J,nil),genexact(C)),P):
                                PrintAllDefinitionsfromRestExt(var(T,X),C,P)
                                PrintAllDefinitionsfromRest(channel(var(T,X)),P)
        'rule' PrintAllDefinitionsfromRestInt(var(T,X), if_then_else(cons(Cons),P,Q),P1):
                                countifinner<- 0
                                PrintAllDefinitionsfromRestif(var(T,X),cons(Cons),P)
                                Nl PrintAllDefinitionsfromRestOtherwise(var(T,X),cons(Cons),P,Q)
                                Nl


/*----------------------------PRINT ALL DEFINITIONS FROM REST IF STATEMENT-------------------------------------------------*/

'action' PrintAllDefinitionsfromRestif(C:VAR, B:BOOLEAN_INT_COMP, P:PROCESS)
        'rule' PrintAllDefinitionsfromRestif(var(T,X),cons(Cons), zero)
        'rule' PrintAllDefinitionsfromRestif(var(T,X),cons(Cons), dotext(E,P)):
                                (|ContainsOtherInstancesOfDefinition(var(T,X),dotext(E,P))
                                Put("if{")
                                Nl Put("[]")
                                Put("@T(")
                                Printcons(Cons)
                                Put(") -> ")
                                VarConsList->L
                                ifc->M1
                                PushConstraintIfVar(cons(Cons),L)
                                ifc->M
                                PrintAllDefinitionsfromRestExtIf(var(T,X),E,P)
                                PrintAllDefinitionsfromRest(channel(var(T,X)),P)|| |)
        'rule' PrintAllDefinitionsfromRestif(var(T,X),cons(Cons), dotint(C,P)):
                                (|ContainsOtherInstancesOfDefinition(var(T,X),dotint(C,P))
                                Put("if{")
                                Nl Put("[]")
```

```
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                    VarConsList->L
                                    PushConstraintIfVar(cons(Cons),L)
                                    PrintAllDefinitionsfromRestInt(var(T,X),C,P) || |)
        'rule' PrintAllDefinitionsfromRestif(var(T,X),cons(Cons), parproc(P,Q))


/*--------------------------PRINT ALL DEFINITIONS FROM REST OTHERWISE-----------------------------------------------------*/

'action' PrintAllDefinitionsfromRestOtherwise(C:VAR, B:BOOLEAN_INT_COMP, Q: PROCESS, P:PROCESS)
        'rule' PrintAllDefinitionsfromRestOtherwise(var(T,X),cons(Cons),P, zero):
                                    Nl (|ContainsOtherInstancesOfDefinition(var(T,X),P)
                                    Put("}")|||)
        'rule' PrintAllDefinitionsfromRestOtherwise(var(T,X),cons(Cons),Q, dotext(E,P)):
                                    Nl (| eq(Q,zero)
                                    ContainsOtherInstancesOfDefinition(var(T,X),dotext(E,P))
                                    Put("if{")
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                    Nl  Put("otherwise ->")
                                    PrintAllDefinitionsfromRestExtOtherwise(var(T,X),E,P)
                                    PrintAllDefinitionsfromRest(channel(var(T,X)),P)
                                    Put("}")
                                    || ContainsOtherInstancesOfDefinition(var(T,X),Q)
                                    ContainsOtherInstancesOfDefinition(var(T,X),dotext(E,P))
                                    Put("otherwise ->")
                                    PrintAllDefinitionsfromRestExtOtherwise(var(T,X),E,P)
                                    PrintAllDefinitionsfromRest(channel(var(T,X)),P)
                                    Put("}")
                                    || ContainsOtherInstancesOfDefinition(var(T,X),dotext(E,P))
                                    Put("if{")
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                    Nl Put("otherwise ->")
                                    PrintAllDefinitionsfromRestExtOtherwise(var(T,X),E,P)
                                    PrintAllDefinitionsfromRest(channel(var(T,X)),P)
                                    Put("}")
                                    || ContainsOtherInstancesOfDefinition(var(T,X),Q) Put("}")
                                    || |)
                                    VarConsList->L
                                    PushConstraintOtherwiseVar(cons(Cons),L)
        'rule' PrintAllDefinitionsfromRestOtherwise(var(T,X),cons(Cons),Q, dotint(C,P)):
                                    Nl (| eq(Q,zero)
                                    ContainsOtherInstancesOfDefinition(var(T,X),dotint(C,P))
                                    Put("if{")
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                    Put("otherwise ->")
                                    PrintAllDefinitionsfromRestInt(var(T,X),C,P)
                                    Put("}")
                                    || ContainsOtherInstancesOfDefinition(var(T,X),Q)
                                    ContainsOtherInstancesOfDefinition(var(T,X),dotint(C,P))
                                    Put("otherwise ->")
                                    PrintAllDefinitionsfromRestInt(var(T,X),C,P)
                                    Put("}")
                                    || ContainsOtherInstancesOfDefinition(var(T,X),dotint(C,P))
                                    Put("if{")
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                    Put("otherwise ->")
                                    PrintAllDefinitionsfromRestInt(var(T,X),C,P)
                                    Put("}")
```

```
                                    || ContainsOtherInstancesOfDefinition(var(T,X),Q)
                                    Put("}")    |)
                                    VarConsList->L
                                    PushConstraintOtherwiseVar(cons(Cons),L)


/*-------------------------INSERT VARIABLE-----------------------------------------------------------------------------*/

'action' InsertVar(V: VAR)
        'rule' InsertVar(var(type(T),X)):
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put("=")
                                    Put(" initially")
                                    id_to_string(T->T1)
                                    (| eq(T1,"integer")
                                    Put(" ")
                                    PutI(0)
                                    Put(" ")
                                    || eq(T1,"real")
                                    Put("0.0")
                                    || Put(" ")
                                    Put("null")
                                    Put(" ") |)
                                    Put(" then ")
                                    Nl


/*-------------------------INSERT @C-----------------------------------------------------------------------------------*/

'action'  InsertatC(C: CHANNEL)
        'rule' InsertatC(channel(var(type(T),X))):
                                    Nl (| id_to_string(T->T1)
                                    ne(T1,"aldef")
                                    Put("if {")
                                    Nl Put("[]")
                                    Put(" ")
                                    Put("@C(")
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put(")")
                                    Put(" ")
                                    Put("->")
                                    Nl ||
                                    Nl Put("[]")
                                    Put(" ")
                                    Put("\"true\"")
                                    Put(" ")
                                    Put("->")
                                    Nl |)


/*-----------------------------------INSERT @C VARIABLE-------------------------------------------------------------*/

'action'  InsertatCV(C: VAR)
        'rule' InsertatCV(var(type(T),X)):
                                    (| id_to_string(T->T1)
                                    ne(T1,"aldef")
                                    Nl Put("if {")
                                    Nl Put("[]")
                                    Put(" ")
                                    Put("@C(")
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put(")")
                                    Put(" ")
                                    Put("->")
                                    Nl || Put("if {")
                                    Nl Put("[]")
                                    Put(" ")
                                    Put("\"true\"")
                                    Put(" ")
```

```
                                        Put("->")
                                        Nl |)

/*------------------------INSERT @T-----------------------------------------------------------------------------------------*/

'action' Insertattrue(C:CHANNEL)
        'rule' Insertattrue(channel(var(T,X))):
                                        Put("if{")
                                        Nl Put("[] \"true\" ->")


/*-----------------------------------PRINT VAR EQ-----------------------------------------------------------------------*/

'action' PrintVarEq(V: VAR,C: CHANNEL)
        'rule' PrintVarEq(var(type(T),X),channel(var(type(T1),Y))):
                                        id_to_string(Y->Y1)
                                        Put(" ")
                                        Put(Y1)
                                         Nl


/*-----------------------------------PRINT VAR CHAN EQ------------------------------------------------------------------*/

'action' PrintVarChanEq(C: CHANNEL,V: VAR)
        'rule' PrintVarChanEq(channel(var(type(T1),Y)),var(type(T),X)):
                                        Nl id_to_string(X->X1)
                                        id_to_string(Y->Y1)
                                        Put(Y1)
                                        Put("= initially")
                                        id_to_string(T1->T2)
                                        (| eq(T2,"integer")
                                        Put(" ")
                                        PutI(0)
                                        Put(" ")
                                        || eq(T2,"real")
                                        Put("0.0")
                                        || Put(" ")
                                        Put("null")
                                        Put(" ") |)
                                        Put(" then if{")
                                        Nl (| id_to_string(T->T3)
                                        ne(T3,"aldef")
                                        Put("[]")
                                        Put("@C(")
                                        Put(X1)
                                        Put(") ->")
                                        Put(X1)
                                        Nl ||
                                        Nl Put("[]")
                                        Put(" ")
                                        Put("\"true\"")
                                        Put(" ")
                                        Put("->")
                                        Put(X1)
                                        Nl |)


/*-----------------------------------PRINT VAR CHAN EQ OTHERWISE--------------------------------------------------------*/

'action' PrintVarChanEqOtherwise(C: CHANNEL,V: VAR,D: BOOLEAN_INT_COMP)
        'rule' PrintVarChanEqOtherwise(channel(var(type(T1),Y)),var(type(T),X), cons(Cons)):
                                        Nl id_to_string(X->X1)
                                        id_to_string(Y->Y1)
                                        Put(Y1)
                                        Put("= initially")
                                        id_to_string(T1->T2)
                                        (| eq(T2,"integer")
                                        Put(" ")
                                        PutI(0)
                                        Put(" ")
                                        || eq(T2,"real")
```

```
                                        Put("0.0")
                                        || Put(" ")
                                        Put("null")
                                        Put(" ") |)
                                        Put(" then if{")
                                        Nl (| ifc->L1
                                        ne(L1,nil)
                                        explorestackif ||  |)
                                        Nl Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Nl Put("otherwise -> if{")
                                        Nl  (| id_to_string(T->T3)
                                        ne(T3,"aldef")
                                        Put("[] @C(")
                                        Put(X1)
                                        Put(") ->")
                                        Put(X1)
                                        Put("")
                                        Nl Put("}")
                                        Nl ||
                                        Nl Put("[]")
                                        Put(" ")
                                        Put("\"true\"")
                                        Put(" ")
                                        Put("->")
                                        Put(X1)
                                        Put("}")
                                        Nl |)


/*---------------------------------PRINT CHAN CONSTANT EQ-------------------------------------------------------------------*/

'action' PrintChanConstantEq(C: CHANNEL,U: IDENT)
                'rule' PrintChanConstantEq(channel(var(type(T1),Y)),U):
                                        id_to_string(U->X1)
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
                                        Put("= initially")
                                         id_to_string(T1->T2)
                                         (| eq(T2,"integer")
                                         Put(" ")
                                         PutI(0)
                                         Put(" ")
                                         || eq(T2,"real")
                                         Put("0.0")
                                         || Put(" ")
                                         Put("null")
                                         Put(" ") |)
                                         Put(" then if{")
                                        Nl (| ifc->L
                                        ne(L,nil)
                                        explorestackif Nl
                                        Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"") Nl
                                        || Nl  Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                         Nl |)


/*---------------------------------PRINT CHAN CONSTANT EQ OTHERWISE---------------------------------------------------------*/

'action' PrintChanConstantEqOtherwise(C: CHANNEL,U: IDENT, D: BOOLEAN_INT_COMP)
                'rule' PrintChanConstantEqOtherwise(channel(var(type(T1),Y)),U, cons(Cons)):
                                        id_to_string(U->X1)
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
```

```
                                        Put("= initially")
                                         id_to_string(T1->T2)
                                         (| eq(T2,"integer")
                                         Put(" ")
                                         PutI(0)
                                         Put(" ")
                                         || eq(T2,"real")
                                         Put("0.0")
                                         || Put(" ")
                                         Put("null")
                                         Put(" ") |)
                                         Put(" then if{")
                                        Nl (| ifc->L
                                        ne(L,nil)
                                        explorestackif Nl
                                        Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Put("if{")
                                        Nl Put("}")
                                        Nl Put("otherwise ->")
                                        Put("if{")
                                        Nl  Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Nl Put("}")
                                         Nl ||
                                         Nl  Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Nl
                                        Nl Put("otherwise ->")
                                        Put("if{")
                                        Nl Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Nl Put("}")
                                         Nl |)


/*-----------------------------------PRINT CHAN INT EQ-----------------------------------------------------------------------*/

'action' PrintChanIntEq(C: CHANNEL,U: INT)
                'rule' PrintChanIntEq(channel(var(type(T1),Y)),U):
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
                                        Put("= initially")
                                         id_to_string(T1->T2)
                                         (| eq(T2,"integer")
                                         Put(" ")
                                         PutI(0)
                                         Put(" ")
                                         || eq(T2,"real")
                                         Put("0.0")
                                         || Put(" ")
                                         Put("null")
                                         Put(" ") |)
                                         Put(" then if{")
                                        Nl Put("[] \"true\" -> ")
                                        PutI(U)
                                         Nl


/*-----------------------------------PRINT CHAN INT EQ OTHERWISE-------------------------------------------------------------*/

'action' PrintChanIntEqOtherwise(C: CHANNEL,U: INT, B:BOOLEAN_INT_COMP)
                'rule' PrintChanIntEqOtherwise(channel(var(type(T1),Y)),U, cons(Cons)):
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
                                        Put("= initially")
                                         id_to_string(T1->T2)
```

```
                                        (| eq(T2,"integer")
                                        Put(" ")
                                        PutI(0)
                                        Put(" ")
                                        || eq(T2,"real")
                                        Put("0.0")
                                        || Put(" ")
                                        Put("null")
                                        Put(" ") |)
                                        Put(" then if{")
                                        Nl (| ifc->L
                                        ne(L,nil)
                                        explorestackif
                                        Nl  Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Put("if{")
                                        Nl Put("}")
                                        Nl Put("otherwise -> if{")
                                        Nl Put("[] \"true\" -> ")
                                        PutI(U)
                                        Nl Put("}")
                                        Nl
                                        || Nl  Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Put("if{")
                                        Nl Put("}")
                                        Nl Put("otherwise -> if{")
                                        Nl Put("[] \"true\" -> ")
                                        PutI(U)
                                        Nl Put("}")
                                        Nl |)


/*----------------------------------PRINT CHAN CONSTANT EQ IF-------------------------------------------------------------------*/

'action' PrintChanConstantEqif(C: CHANNEL,U: IDENT,B:BOOLEAN_INT_COMP)
                'rule' PrintChanConstantEqif(channel(var(type(T1),Y)),U,cons(Cons)):
                                        id_to_string(U->X1)
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
                                        Put("= initially")
                                        id_to_string(T1->T2)
                                        (| eq(T2,"integer")
                                        Put(" ")
                                        PutI(0)
                                        Put(" ")
                                        || eq(T2,"real")
                                        Put("0.0")
                                        || Put(" ")
                                        Put("null")
                                        Put(" ") |)
                                        Put(" then if{")
                                        Nl
                                        Nl  (| ifc->L
                                        ne(L,nil)
                                        explorestackif
                                        Nl Put("[] @T(")
                                        Printcons(Cons)
                                        Put(")")
                                        Put(" -> ")
                                        Put("if{")
                                        Nl Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put("}")
                                        Nl
                                        || Nl Put("[] @T(")
                                        Printcons(Cons)
                                        Put(")")
                                        Put(" -> ")
```

```
                                        Put("if{")
                                        Nl Put("[] \"true\" ->")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put("}")
                                        Nl |)


/*----------------------------------PRINT CHAN INT EQ IF-------------------------------------------------------------------*/

'action' PrintChanIntEqif(C: CHANNEL,U: INT, B:BOOLEAN_INT_COMP)
            'rule' PrintChanIntEqif(channel(var(type(T1),Y)),U, cons(Cons)):
                                        id_to_string(Y->Y1)
                                        Nl Put(Y1)
                                        Put("= initially")
                                         id_to_string(T1->T2)
                                         (| eq(T2,"integer")
                                         Put(" ")
                                         PutI(0)
                                         Put(" ")
                                         || eq(T2,"real")
                                         Put("0.0")
                                         || Put(" ")
                                         Put("null")
                                         Put(" ") |)
                                         Put(" then if{")
                                        Nl  (| ifc->L
                                        ne(L,nil)
                                        explorestackif
                                        Put("[] @T(")
                                         Printcons(Cons)
                                         Put(") ->")
                                         Nl PutI(U)
                                         Nl
                                        || Put("[] @T(")
                                         Printcons(Cons)
                                         Put(") ->")
                                         Nl PutI(U)
                                         Nl |)


/*----------------------------------PRINT VAN CHAN EQ IF-------------------------------------------------------------------*/

'action' PrintVarChanEqif(C: CHANNEL,V: VAR,B:BOOLEAN_INT_COMP)
            'rule' PrintVarChanEqif(channel(var(type(T1),Y)),var(type(T),X),cons(Cons)):
                                        Nl id_to_string(X->X1)
                                        id_to_string(Y->Y1)
                                        Put(Y1)
                                        Put("= initially")
                                         id_to_string(T1->T2)
                                         (| eq(T2,"integer")
                                         Put(" ")
                                         PutI(0)
                                         Put(" ")
                                         || eq(T2,"real")
                                         Put("0.0")
                                         || Put(" ")
                                         Put("null")
                                         Put(" ") |)
                                         Put(" then if{")
                                        Nl  (| ifc->L
                                        ne(L,nil)
                                        explorestackif
                                        Nl Put("[]")
                                        Put("@T(")
                                        Printcons(Cons)
                                        Put(") ->")
                                        Put("if{")
                                        Nl (| id_to_string(T->T3)
                                         ne(T3,"aldef")
                                         Put("[]")
                                        Put("@C(")
```

```
                                    Put(X1)
                                    Put(") ->")
                                    Put(X1)
                                     Nl  Put("}")
                                     Nl ||
                                    Nl Put("[]")
                                    Put(" ")
                                     Put("\"true\"")
                                     Put(" ")
                                    Put("->")
                                     Put(X1)
                                     Put("}")
                                     Nl |)
                                    || Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") ->")
                                    Put("if{")
                                    Nl (| id_to_string(T->T3)
                                     ne(T3,"aldef")
                                     Put("[]")
                                    Put("@C(")
                                    Put(X1)
                                    Put(") ->")
                                    Put(X1)
                                     Put("}")
                                     Nl ||
                                    Nl Put("[]")
                                    Put(" ")
                                     Put("\"true\"")
                                     Put(" ")
                                    Put("->")
                                     Put(X1)
                                     Put("}")
                                     Nl |) |)


/*-------------------------------------PRINT CHANNEL-------------------------------------------------------------------------*/

'action'  PrintC(C: CHANNEL)
          'rule' PrintC(channel(var(T,X))):
                                    id_to_string(X->X1) Put(X1)


/*-------------------------------------@C VARIABLE LIST-----------------------------------------------------------------------*/

'action' atCvarlist(V: VARLIST)
          'rule' atCvarlist(nil):
                                     Put(" -> ")
          'rule' atCvarlist(varlist(var(type(T),X),L)):
                                     (|id_to_string(T->T1)
                                     ne(T1,"aldef")
                                     Put("@C(")
                                     PrintvariableNoType(var(type(T),X))
                                     Put(")")
                                     (|ne(L,nil)
                                     ContainsPrintableType(L)
                                     Put(" & ")
                                     || |)
                                     || |)
                                     atCvarlist(L)


/*-----------------------------------PRINT REST SERVICE CALLS-------------------------------------------------------------*/

'action' PrintRestServiceCalls(C:INT_COMP,P:PROCESS)
          'rule' PrintRestServiceCalls(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P):
                                    Put("[]")
                                     id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
```

```
                        Put(I1)
                        Put(":")
                        Put(J1)
                        Put("(\"")
                        Put(" + ")
                        PrintVarlist(varlist(V,L))
                        Put("\"")
                         Put(X1)
                         Put("\"")
                        Put(" + ")
                        Put("\")\"")
                        Put("")
                        Nl PrintRestServiceCalls(C,P)
'rule' PrintRestServiceCalls(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P):
                        Put("[]")
                        id_to_string(X->X1)
                        atCvarlist(varlist(V,L))
                        id_to_string(I->I1)
                        id_to_string(J->J1)
                        Put("\"")
                        Put(I1)
                        Put(":")
                        Put(J1)
                        Put("(\"")
                        Put(" + ")
                        PrintVarlist(varlist(V,L))
                        Put("\"")
                         Put(X1)
                         Put("\"")
                        Put(" + ")
                        Put("\")\"")
                        Put("")
                        Nl
                         PrintAllServicesFromRest(P)
'rule' PrintRestServiceCalls(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P):
                        Put("[]")
                        id_to_string(X->X1)
                         Put(" \"true\" -> ")
                        id_to_string(I->I1)
                        id_to_string(J->J1)
                        Put("\"")
                        Put(I1)
                        Put(":")
                        Put(J1)
                        Put("(\"")
                        Put("+")
                        Put("\"")
                         Put(X1)
                         Put("\"")
                        Put("+")
                        Put("\")\"")
                        Put("")
                        Nl PrintRestServiceCalls(C,P)
'rule' PrintRestServiceCalls(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P):
                        Put("[]")
                        id_to_string(X->X1)
                         Put(" \"true\" -> ")
                        id_to_string(I->I1)
                        id_to_string(J->J1)
                        Put("\"")
                        Put(I1)
                        Put(":")
                        Put(J1)
                        Put("(\"")
                        Put("+")
                        Put("\"")
                         Put(X1)
                         Put("\"")
                        Put("+")
                        Put("\")\"")
                        Put("")
                        Nl
                         PrintAllServicesFromRest(P)
```

```
        'rule' PrintRestServiceCalls(if_then_else(cons(Cons),P,Q),P1):
                                PrintAllServicesFromRestIf(cons(Cons),P)
                                PrintAllServicesFromRestOtherwise(cons(Cons),P,Q)
                                Nl

/*---------------------------------------PRINT ALL SERVICES FROM REST-----------------------------------------------------*/

'action' PrintAllServicesFromRest(P:PROCESS)
        'rule'   PrintAllServicesFromRest(zero)
        'rule'   PrintAllServicesFromRest(dotext(E,P)):
                                PrintAllServicesFromRest(P)
        'rule'   PrintAllServicesFromRest(dotint(C1,P)):
                                PrintRestServiceCalls(C1,P)



/*-------------------------------------------PRINT ALL SERVICES FROM REST IF-----------------------------------------------*/

'action' PrintAllServicesFromRestIf(C:BOOLEAN_INT_COMP, P: PROCESS)
        'rule'  PrintAllServicesFromRestIf(cons(Cons),zero)
        'rule'  PrintAllServicesFromRestIf(cons(Cons),dotext(E,P)):
                                (| ContainsServices(dotext(E,P))
                                 Put("[] \"true\" ->")
                                 Put("if{")
                                Nl Put("[] @T(")
                                Printcons(Cons)
                                Put(") -> ")
                                Put("if{")
                                 Nl  PrintAllServicesFromRest(dotext(E,P))
                                 Put("}")
                                 || |)
          'rule'  PrintAllServicesFromRestIf(cons(Cons),dotint(C,P)):
                                (|ContainsServices(dotint(C,P))
                                 Put("[] \"true\" ->")
                                 Put("if{")
                                Nl
                                Put("[] @T(")
                                Printcons(Cons)
                                Put(") -> ")
                                Put("if{")
                                 Nl  PrintAllServicesFromRest(dotint(C,P))
                                 Put("}")|| |)



/*-------------------------------------------PRINT ALL SERVICES FROM REST OTHERWISE----------------------------------------*/

'action' PrintAllServicesFromRestOtherwise(C:BOOLEAN_INT_COMP, P: PROCESS,Q: PROCESS)
        'rule'  PrintAllServicesFromRestOtherwise(cons(Cons),Q,zero): Nl (|ContainsServices(Q) Put("}") || |)
        'rule'  PrintAllServicesFromRestOtherwise(cons(Cons),Q,dotext(E,P)):
                                Nl (leq(Q,zero)
                                 ContainsServices(dotext(E,P))
                                 Put("if{")
                                 Nl Put("[]")
                                Put("@T(")
                                Printcons(Cons)
                                Put(") -> ")
                                 Put("otherwise ->")
                                 Put("if{")
                                 Nl  PrintAllServicesFromRest(dotext(E,P))
                                 Put("}")
                                 Nl  Put("}")
                                 || ContainsServices(Q)
                                 ContainsServices(dotext(E,P))
                                 Put("otherwise ->")
                                 Put("if{")
                                 Nl  PrintAllServicesFromRest(dotext(E,P))
                                 Put("}")
                                 Nl Put("}")
                                 || ContainsServices(dotext(E,P))
                                 Put("[] \"true\" ->")
                                 Put("if{")
                                 Nl Put("[]")
```

```
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                     Put("otherwise ->")
                                     Put("if{")
                                     Nl  PrintAllServicesFromRest(dotext(E,P))
                                     Put("}")
                                     Nl  Put("}")
                                     || ContainsServices(Q)
                                     Put("}") ||  |)
        'rule'  PrintAllServicesFromRestOtherwise(cons(Cons),Q,dotint(C,P)):
                                    Nl (leq(Q,zero)
                                     ContainsServices(dotint(C,P))
                                     Put("if{")
                                     Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                     Put("otherwise ->")
                                     Put("if{")
                                     Nl  PrintAllServicesFromRest(dotint(C,P))
                                     Put("}")
                                     Nl Put("}")
                                     || ContainsServices(Q)
                                     ContainsServices(dotint(C,P))
                                     Put("otherwise ->")
                                     Put("if{")
                                     Nl  PrintAllServicesFromRest(dotint(C,P))
                                     Put("}")
                                     Nl  Put("}")
                                     || ContainsServices(dotint(C,P))
                                     Put("[] \"true\" ->")
                                     Put("if{")
                                     Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(") -> ")
                                     Put("otherwise ->")
                                     Put("if{")
                                     Nl  PrintAllServicesFromRest(dotint(C,P))
                                     Put("}")
                                     Nl Put("}")
                                     || ContainsServices(Q)
                                     Put("}") ||   |)


/*---------------------------------PUSH CONSTRAINT OTHERWISE-----------------------------------------------------------------------*/

'action' PushConstraintOtherwise(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintOtherwise(cons(Cons),nil):
                                    ifc<-conlist(nil,intpair(cons(Cons),1))
        'rule' PushConstraintOtherwise(cons(Cons),conlist(U,V)):
                                    PushConstraintOtherwise(cons(Cons),U)
                                    ifc->L
                                    ifc <- conlist(L,V)



/*---------------------------------PUSH CONSTRAINT OTHERWISE VARIABLE---------------------------------------------------------*/


'action' PushConstraintOtherwiseVar(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintOtherwiseVar(cons(Cons),nil):
                                    VarConsList<-conlist(nil,intpair(cons(Cons),1))
        'rule' PushConstraintOtherwiseVar(cons(Cons),conlist(U,V)):
                                    PushConstraintOtherwiseVar(cons(Cons),U)
                                    VarConsList->L
                                    VarConsList<- conlist(L,V)



/*-------------------------------------PUSH CONSTRAINT OTHERWISE SERVICE-------------------------------------------------*/
```

```
'action' PushConstraintOtherwiseServ(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintOtherwiseServ(cons(Cons),nil):
                        ServConsList<-conlist(nil,intpair(cons(Cons),1))
        'rule' PushConstraintOtherwiseServ(cons(Cons),conlist(U,V)):
                        PushConstraintOtherwiseServ(cons(Cons),U)
                        ServConsList->L
                        ServConsList<- conlist(L,V)


/*---------------------------------PUSH CONSTRAINT IF-------------------------------------------------------------------------*/

'action' PushConstraintIf(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintIf(cons(Cons),nil):
                        ifc<-conlist(nil,intpair(cons(Cons),0))
        'rule' PushConstraintIf(cons(Cons),conlist(U,V)):
                        PushConstraintIf(cons(Cons),U)
                        ifc->L
                        ifc <- conlist(L,V)


/*---------------------------------------PUSH CONSTRAINT IF VARIABLE-----------------------------------------------------------*/

'action' PushConstraintIfVar(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintIfVar(cons(Cons),nil):
                        VarConsList<-conlist(nil,intpair(cons(Cons),0))
        'rule' PushConstraintIfVar(cons(Cons),conlist(U,V)):
                        VarConsList ->L
                        VarConsList <- conlist(L,V)


/*---------------------------------------PUSH CONSTRAINT IF SERVICE-----------------------------------------------------------*/

'action' PushConstraintIfServ(B: BOOLEAN_INT_COMP, L: BOOLEAN_INT_COMP_LIST)
        'rule' PushConstraintIfServ(cons(Cons),nil):
                        ServConsList<-conlist(nil,intpair(cons(Cons),0))
        'rule' PushConstraintIfServ(cons(Cons),conlist(U,V)):
                        ServConsList ->L
                        ServConsList <- conlist(L,V)


/*-----------------------------------------CLOSE BRACKETS-----------------------------------------------------------------*/

'action' CloseBrackets
        'rule' CloseBrackets:
                        (|newstack-> N
                        ifzero(N)
                        || Nl Put("}")
                        Nl newstack -> N
                        newstack <- N-1
                        CloseBrackets |)



/*---------------------------------CLOSE BRACKETS VARIABLE--------------------------------------------------------------------*/

'action' CloseBracketsVar
        'rule' CloseBracketsVar:
                        (|newstackvar-> N
                        ifzero(N)
                        || Nl Put("}")
                        Nl newstackvar -> N
                        newstackvar <- N-1
                        CloseBracketsVar |)



/*---------------------------------------CLOSE BRACKETS SERVICE---------------------------------------------------------------*/

'action' CloseBracketsServ
        'rule' CloseBracketsServ:
                        (|newstackserv-> N
                        ifzero(N)
```

```
                                    || Nl Put("}")
                                    Nl newstackserv -> N
                                    newstackserv <- N-1
                                    CloseBracketsServ |)


/*-------------------------------------------------INSERT VAR IF-----------------------------------------------------------*/

'action' InsertVarIf(V: VAR)
            'rule' InsertVarIf(var(type(T),X)):
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put("= initially")
                                     id_to_string(T->T1)
                                     (| eq(T1,"integer")
                                     Put(" ")
                                     PutI(0)
                                     Put(" ")
                                     || eq(T1,"real")
                                     Put("0.0")
                                     || Put(" ")
                                     Put("null")
                                     Put(" ") |)
                                     Put(" then ")
                                    Nl


/*---------------------------------------INSERT AT C IF-----------------------------------------------------------------*/

'action'  InsertatCif(C: CHANNEL,B:BOOLEAN_INT_COMP)
            'rule' InsertatCif(channel(var(type(T),X)),cons(Cons)):
                                    Nl  Put("if {")
                                    Nl (| ifc->L
                                    ne(L,nil)
                                    explorestackif
                                    Put("[]")
                                    Put(" ")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")->")
                                    Put(" ")
                                    Put("if{")
                                    Nl(| id_to_string(T->T3)
                                     ne(T3,"aldef")
                                     Put("[]")
                                    Put("@C(")
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put(")")
                                    Put(" ")
                                    Put("->")
                                     Nl ||
                                    Nl Put("[]")
                                    Put(" ")
                                     Put("\"true\"")
                                     Put(" ")
                                    Put("->")
                                     Nl |)
                                    || Put("[]")
                                    Put(" ")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")->")
                                    Put(" ")
                                    Put("if{")
                                    Nl(| id_to_string(T->T3)
                                     ne(T3,"aldef")
                                     Put("[]")
                                    Put("@C(")
                                    id_to_string(X->X1)
                                    Put(X1)
                                    Put(")")
```

```
                                    Put(" ")
                                    Put("->") Nl
                                    || Nl Put("[]")
                                    Put(" ")
                                     Put("\"true\"")
                                     Put(" ")
                                    Put("->")
                                     Nl |)|)


/*----------------------------------------EXPLORE STACK IF----------------------------------------------------------------*/

'action' explorestackif
            'rule' explorestackif:
                                     ifc->L
                                     ifc1<-L
                                     explorestackifnext


/*-------------------------------------------EXPLORE STACK IF NEXT-----------------------------------------------------*/


'action' explorestackifnext
            'rule' explorestackifnext:
                                    ifc1->conlist(U,intpair(cons(Cons),B))
                                    (leq(B,0)
                                    ne(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstack-> N
                                    newstack <- N+1
                                    Nl ifc1<-U
                                     explorestackifnext
                                     Nl ||  eq(B,0)
                                    eq(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstack-> N
                                    newstack <- N+1 Nl
                                    || eq(B,1)
                                    ne(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Nl Put("otherwise -> ")
                                    Put("if{")
                                    newstack-> N
                                    newstack <- N+1
                                    Nl ifc1<-U
                                     explorestackifnext
                                    Nl || eq(B,1)
                                    eq(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Nl Put("otherwise -> ")
                                    Put("if{")
                                    newstack-> N
                                    newstack <- N+1
                                     Nl |)


/*----------------------------------------EXPLORE STACK IF VARIABLE-------------------------------------------------------*/

'action' explorestackifVar
```

```
'rule' explorestackifVar:
                                    VarConsList->conlist(U,intpair(cons(Cons),B))
                                    (leq(B,0)
                                    ne(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstackvar-> N
                                    newstackvar <- N+1
                                    Nl explorestackifVar Nl
                                    ||  eq(B,0)
                                    eq(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstackvar-> N
                                    newstackvar <- N+1
                                     Nl
                                    || eq(B,1)
                                    ne(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Nl Put("otherwise -> ")
                                    Put("if{")
                                    newstackvar-> N
                                    newstackvar <- N+1
                                    Nl explorestackifVar
                                    Nl || eq(B,1)
                                    eq(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Nl Put("otherwise -> ")
                                    Put("if{")
                                    newstackvar-> N
                                    newstackvar <- N+1
                                     Nl |)


/*-----------------------------------------------------EXPLORE STACK IF SERVICE-----------------------------------------------------------------
*/

'action' explorestackifServ
            'rule' explorestackifServ:
                                    ServConsList->conlist(U,intpair(cons(Cons),B))
                                    (leq(B,0)
                                    ne(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstackserv-> N
                                    newstackserv <- N+1
                                    Nl explorestackifServ
                                     Nl
                                    ||  eq(B,0)
                                    eq(U,nil)
                                    Put("[] @T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put("  -> ")
                                    Put("if{")
                                    newstackserv-> N
                                    newstackserv <- N+1 Nl
                                    || eq(B,1)
```

```
                                        ne(U,nil)
                                        Put("[] @T(")
                                        Printcons(Cons)
                                        Put(")")
                                        Put("  -> ")
                                        Nl Put("otherwise -> ")
                                        Put("if{")
                                        newstackserv-> N
                                        newstackserv <- N+1
                                        Nl explorestackifServ
                                        Nl || eq(B,1)
                                        eq(U,nil)
                                        Put("[] @T(")
                                        Printcons(Cons)
                                        Put(")")
                                        Put("  -> ")
                                        Nl Put("otherwise -> ")
                                        Put("if{")
                                        newstackserv-> N
                                        newstackserv <- N+1
                                         Nl |)


/*---------------------------------------TRANSLATE INT-------------------------------------------------------------------*/


'action' TranslateInt(C: INT_COMP, P:PROCESS)
        'rule' TranslateInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P):
                                        (|service->M
                                        ifzero(M)
                                        service<-1
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl Put("[]")
                                        id_to_string(X->X1)
                                        atCvarlist(varlist(V,L))
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
                                         Put(X1)
                                         Put("\"")
                                        Put(" + ")
                                        Put("\")\"")
                                        Put("")
                                        Nl PrintRestServiceCalls(C,P)
                                        Nl Put("};")
                                        TranslateGen(genincomp(C),P)
                                        || TranslateGen(genincomp(C),P)|)
        'rule' TranslateInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P):
                                        (|service->M
                                        ifzero(M)
                                        service<-1
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl Put("[]")
                                        id_to_string(X->X1)
                                        atCvarlist(varlist(V,L))
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
```

```
                                    Put(X1)
                                    Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Nl PrintAllServicesFromRest(P)
                                    Put("};")
                                    Nl TranslateGen(genexact(C),P)
                                    || TranslateGen(genexact(C),P)|)
        'rule' TranslateInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P):
                                    (|service->M
                                    ifzero(M)
                                    service<-1
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    Put(" \"true\" ->")
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put("+")
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put("+")
                                    Put("\")\"")
                                    Put("")
                                    Nl PrintRestServiceCalls(C,P)
                                    Nl Put("};")
                                    TranslateGen(genincomp(C),P)
                                    || TranslateGen(genincomp(C),P)|)
        'rule' TranslateInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P):
                                    (|service->M
                                    ifzero(M)
                                    service<-1
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    Put(" \"true\" ->")
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put("+")
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put("+")
                                    Put("\")\"")
                                    Put("")
                                    Nl PrintAllServicesFromRest(P)
                                    Put("};")
                                    Nl TranslateGen(genexact(C),P)
                                    || TranslateGen(genexact(C),P)|)
        'rule' TranslateInt(if_then_else(cons(Cons),P,Q),P1):
                                    TranslateProcessIf(P,Q,cons(Cons))


/*-------------------------------------TRANSLATE PROCESS IF-----------------------------------------------------------*/

'action' TranslateProcessIf(P: PROCESS, Q: PROCESS, B: BOOLEAN_INT_COMP)
        'rule' TranslateProcessIf(zero,zero,cons(Cons))
        'rule' TranslateProcessIf(zero,Q,cons(Cons)):
                                    TranslateOtherwiseProcess(Q,cons(Cons))
        'rule' TranslateProcessIf(P,zero,cons(Cons)):
                                    TranslateifModuleProcess(P,cons(Cons))
```

```
	'rule' TranslateProcessIf(dotext(E,P),Q,cons(Cons)):
					TranslateExtIf(E,P,cons(Cons),Q)
					TranslateProcessIf(P,Q,cons(Cons))
					(| IsStack(0)
					Put("")
					|| Put("}")
					stack->N
					stack<-N-1 |)
	'rule' TranslateProcessIf(dotint(C,P),Q,cons(Cons)):
					TranslateIntIf(C,P,cons(Cons),Q)
					TranslateProcessIf(P,Q,cons(Cons))


/*-----------------------------------TRANSLATE OTHERWISE PROCESS-----------------------------------------------------*/

'action' TranslateOtherwiseProcess(Q: PROCESS, B: BOOLEAN_INT_COMP)
	'rule' TranslateOtherwiseProcess(zero,cons(Cons))
	'rule' TranslateOtherwiseProcess(dotext(E,P),cons(Cons)):
					TranslateExtOtherwiseModule(E,P,cons(Cons))
					 TranslateOtherwiseProcess(P,cons(Cons))
					(| IsStack(0)
					|| Put("}")
					stack->N
					stack<-N-1 |)
	'rule' TranslateOtherwiseProcess(dotint(C,P),cons(Cons)):
					TranslateIntOtherwiseModule(C,P,cons(Cons))



/*-----------------------------------TRANSLATE EXT OTHERWISE MODULE--------------------------------------------------*/

'action' TranslateExtOtherwiseModule(E: EXT_ACT, P:PROCESS, B: BOOLEAN_INT_COMP)
	'rule' TranslateExtOtherwiseModule(comm(bracket(C,V)), P, cons(Cons)):
					Nl InsertVar(V)
					Put("if{") Nl
					(| ifc->L
					ne(L,nil)
					explorestackif
					Nl Put("[] @T(")
					Printcons(Cons)
					Put(") -> ")
					Nl Put("otherwise ->")
					Put("if{")
					Nl Put("[]")
					 Put(" @C(")
					PrintC(C)Put(") -> ")
					Nl PrintVarEq(V,C)
					Put("")
					 Put("")
					Nl Put("}")
					CloseBrackets
					 Put("}")
					 Put(";")
					 Nl
					|| Nl Put("[] @T(")
					Printcons(Cons)
					Put(") -> ")
					Nl Put("otherwise ->")
					Put("if{")
					Nl Put("[]")
					 Put(" @C(")
					PrintC(C)
					 Put(") -> ")
					Nl
					PrintVarEq(V,C)
					Put(" ")
					 Put("")
					Nl Put("}")
					Nl Put("};")
					 Nl |)
	'rule' TranslateExtOtherwiseModule(comm(angle(C,V)), P, cons(Cons)):
					(|ChecknotVar(C)
					checkl->L
```

```
                                checkl<-lst(varpair(C,1),L)
                                PrintVarChanEqOtherwise(C,V,cons(Cons))
                                PrintAllDefinitionsfromRest(C,P)
                                Nl Put("}")
                                 CloseBrackets Put(";")
                                 || |)
        'rule' TranslateExtOtherwiseModule(comm(send(C,U)), P, cons(Cons)):
                                (|ChecknotVar(C)
                                ifc->L1
                                ne(L1,nil)
                                explorestackif
                                checkl->L
                                checkl<- lst(varpair(C,1),L)
                                PrintChanConstantEqOtherwise(C,U,cons(Cons))
                                PrintAllDefinitionsfromRest(C,P)
                                Nl Put("}")
                                 CloseBrackets Put(";")
                                 || ChecknotVar(C)
                                checkl->L
                                checkl<- lst(varpair(C,1),L)
                                PrintChanConstantEqOtherwise(C,U,cons(Cons))
                                PrintAllDefinitionsfromRest(C,P)
                                Nl Put("}")
                                 CloseBrackets
                                 Put(";") ||  |)
        'rule' TranslateExtOtherwiseModule(comm(send1(C,U)), P, cons(Cons)):
                                (|ChecknotVar(C)
                                ifc->L1
                                ne(L1,nil)
                                explorestackif
                                checkl->L
                                checkl<- lst(varpair(C,1),L)
                                PrintChanIntEqOtherwise(C,U,cons(Cons))
                                PrintAllDefinitionsfromRest(C,P)
                                Nl Put("}")
                                 CloseBrackets
                                 Put(";")
                                 || ChecknotVar(C)
                                checkl->L
                                checkl<- lst(varpair(C,1),L)
                                PrintChanIntEqOtherwise(C,U,cons(Cons))
                                PrintAllDefinitionsfromRest(C,P)
                                Nl Put("}")
                                 CloseBrackets
                                 Put(";")||  |)



/*---------------------------------TRANSLATE INT OTHERWISE MODULE-----------------------------------------------------------*/

'action' TranslateIntOtherwiseModule(C: INT_COMP, P: PROCESS, B:BOOLEAN_INT_COMP)
        'rule' TranslateIntOtherwiseModule(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)), P,
cons(Cons)):
                                (|service->M
                                ifzero(M)
                                ifc->L1 ne(L1,nil)
                                service<-1
                                Nl Put("cServiceInvoke = initially null then if {")
                                Nl explorestackif
                                Nl  Put("[] @T(")
                                Printcons(Cons)
                                Put(") -> ")
                                Nl Put("otherwise ->")
                                Put("if{")
                                Nl  Put("[]")
                                 atCvarlist(varlist(V,L))
                                id_to_string(X->X1)
                                id_to_string(I->I1)
                                id_to_string(J->J1)
                                Put("\"")
                                Put(I1)
                                Put(":")
```

```
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
                                         Put(X1)
                                         Put("\"")
                                        Put(" + ")
                                        Put("\")\"")
                                        Put("")
                                        Nl
                                        Nl PrintRestServiceCalls(C,P)
                                        Nl Put("}")
                                        Nl Put("}")
                                        CloseBrackets
                                         Put(";")
                                         ifc->L2
                                         PushConstraintOtherwise(cons(Cons),L2)
                                        TranslateGen(genincomp(C),P)
                                        || service->M
                                        ifzero(M)
                                        service<-1
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl  Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Nl Put("otherwise ->")
                                        Put("if{")
                                        Nl  Put("[]")
                                         atCvarlist(varlist(V,L))
                                        id_to_string(X->X1)
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
                                         Put(X1)
                                         Put("\"")
                                        Put(" + ")
                                        Put("\")\"")
                                        Put("")
                                        Nl
                                        Nl PrintRestServiceCalls(C,P)
                                         Nl Put("}")
                                        Nl Put("}")
                                        CloseBrackets Put(";")
                                         ifc->L2
                                         PushConstraintOtherwise(cons(Cons),L2)
                                        TranslateGen(genincomp(C),P)
                                        || ifc->L2
                                         PushConstraintOtherwise(cons(Cons),L2)
                                         TranslateGen(genincomp(C),P)|)
'rule' TranslateIntOtherwiseModule(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)), P, cons(Cons)):
                                        (|service->M ifzero(M)
                                         ifc->L1 ne(L1,nil)
                                         service<-1
                                         Nl Put("cServiceInvoke = initially null then if {")
                                         Nl explorestackif
                                         Nl  Put("[] @T(")
                                         Printcons(Cons)
                                         Put(") -> ")
                                         Nl Put("otherwise ->")
                                         Put("if{")
                                         Nl  Put("[]")
                                         atCvarlist(varlist(V,L))
                                         id_to_string(X->X1)
                                         id_to_string(I->I1)
                                         id_to_string(J->J1)
```

```
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put(" + ")
                                        Put("\")\"")
                                        Put("") Nl
                                        Nl PrintAllServicesFromRest(P)
                                        Nl Put("}")
                                        Nl   Put("}")
                                        CloseBrackets
                                        Put(";")
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P)
                                        || service->M
                                        ifzero(M)
                                        service<-1
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl
                                        Nl  Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") -> ")
                                        Nl Put("otherwise ->")
                                        Put("if{")
                                        Nl  Put("[]")
                                        atCvarlist(varlist(V,L))
                                        id_to_string(X->X1)
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put(" + ")
                                        PrintVarlist(varlist(V,L))
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put(" + ")
                                        Put("\")\"")
                                        Put("")
                                        Nl
                                        Nl PrintAllServicesFromRest(P)
                                        Nl Put("}")
                                        Nl Put("}")
                                        CloseBrackets
                                        Put(";")
                                        ifc->L2
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P)
                                        || ifc->L2
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P)              |)
'rule' TranslateIntOtherwiseModule(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)), P, cons(Cons)):
                                       (|service->M
                                        ifzero(M)
                                        service<-1
                                        ifc->L1
                                        ne(L1,nil)
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl explorestackif
                                        Nl Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") ->")
                                        Nl  Put("otherwise ->")
                                        Put("if{")
                                        Nl Put("[]")
                                        Put(" \"true\" ")
                                        id_to_string(X->X1)
```

```
                                id_to_string(I->I1)
                                id_to_string(J->J1)
                                Put("\"")
                                Put(I1)
                                Put(":")
                                Put(J1)
                                Put("(\"")
                                Put("+")
                                Put("\"")
                                Put(X1)
                                Put("\"")
                                Put("+")
                                Put("\")\"")
                                Put("")
                                Nl PrintRestServiceCalls(C,P)
                                Nl Put("}")
                                CloseBrackets
                                Put(";")
                                TranslateGen(genincomp(C),P)
                                || service->M
                                ifzero(M)
                                service<-1
                                Nl Put("cServiceInvoke = initially null then if {")
                                Nl
                                Nl   Put("[] @T(")
                                Printcons(Cons)
                                Put(") ->")
                                Nl  Put("otherwise ->")
                                Put("if{")
                                Nl Put("[]")
                                Put(" \"true\" ")
                                id_to_string(X->X1)
                                id_to_string(I->I1)
                                id_to_string(J->J1)
                                Put("\"")
                                Put(I1)
                                Put(":")
                                Put(J1)
                                Put("(\"")
                                Put("+")
                                Put("\"")
                                Put(X1)
                                Put("\"")
                                Put("+")
                                Put("\")\"")
                                Put("")
                                Nl PrintRestServiceCalls(C,P)
                                Nl Put("}")
                                CloseBrackets
                                Put(";")
                                ifc->L2
                                PushConstraintOtherwise(cons(Cons),L2)
                                TranslateGen(genincomp(C),P)
                                || ifc->L2
                                PushConstraintOtherwise(cons(Cons),L2)
                                TranslateGen(genincomp(C),P)|)
'rule' TranslateIntOtherwiseModule(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)), P, cons(Cons)):
                                (|service->M
                                ifzero(M)
                                service<-1
                                Nl ifc->L1
                                ne(L1,nil)
                                Put("cServiceInvoke = initially null then if {")
                                Nl explorestackif
                                Nl Put("[] @T(")
                                Printcons(Cons)
                                Put(") ->")
                                Nl Put("otherwise ->")
                                Put("if{")
                                Nl Put("[]")
                                Put(" \"true\" ")
                                id_to_string(X->X1)
                                id_to_string(I->I1)
```

```
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put("+")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put("+")
                                        Put("\")\"")
                                        Put("")
                                        Nl PrintAllServicesFromRest(P)
                                        Nl  Put("}")
                                        CloseBrackets
                                        Put(";")
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P)
                                        || service->M
                                        ifzero(M)
                                        service<-1
                                        Nl Put("cServiceInvoke = initially null then if {")
                                        Nl
                                        Nl   Put("[] @T(")
                                        Printcons(Cons)
                                        Put(") ->")
                                        Nl  Put("otherwise ->")
                                        Put("if{")
                                        Nl Put("[]")
                                        Put(" \"true\" ")
                                        id_to_string(X->X1)
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put("+")
                                        Put("\"")
                                        Put(X1)
                                        Put("\"")
                                        Put("+")
                                        Put("\")\"")
                                         Put("")
                                        Nl  PrintAllServicesFromRest(P)
                                        Nl Put("}")
                                        CloseBrackets Put(";")
                                        ifc->L2
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P)
                                        || ifc->L2
                                        TranslateGenConsOth(genexact(C),zero,cons(Cons),P) |)
        'rule' TranslateIntOtherwiseModule(if_then_else(cons(Cons),P,Q), P1, cons(Cons1)):
                                        ifc->L
                                        PushConstraintOtherwise(cons(Cons1),L)
                                        TranslateProcessIf(P,Q,cons(Cons))



/*-------------------------------------TRANSLATE IF MODULE PROCESS-----------------------------------------------------------*/

'action' TranslateifModuleProcess(P:PROCESS, B:BOOLEAN_INT_COMP)
        'rule' TranslateifModuleProcess(zero,cons(Cons))
        'rule' TranslateifModuleProcess(dotext(E,P),cons(Cons)):
                                        TranslateExtIf(E,P,cons(Cons),zero)
                                         TranslateifModuleProcess(P,cons(Cons))
        'rule' TranslateifModuleProcess(dotint(C,P),cons(Cons)):
                                        TranslateIntIf(C,P,cons(Cons),zero)

/*-----------------------------------TRANSLATE EXT IF-------------------------------------------------------------------------*/

'action' TranslateExtIf(E: EXT_ACT, P:PROCESS, B:BOOLEAN_INT_COMP,Q:PROCESS)
        'rule' TranslateExtIf(comm(bracket(C,V)),P,cons(Cons),Q):
```

```
                                    Nl   InsertVarIf(V)
                                    InsertatCif(C,cons(Cons))
                                    PrintVarEq(V,C)
                                    Put("}")
                                    Nl
                                    Put("}") Nl
                                    CloseBrackets Put(";")
                                     Nl
                                     Nl
        'rule' TranslateExtIf(comm(angle(C,V)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintVarChanEqif(C,V,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q)|| |)
                                    Nl CloseBrackets
                                     Put("}")
                                     Put(";")
                                     Nl
                                     Nl  || |)
        'rule' TranslateExtIf(comm(send(C,U)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintChanConstantEqif(C,U,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q) || |)
                                    Put("}")
                                    CloseBrackets
                                     Put(";")
                                     Nl
                                     Nl ||  |)
        'rule' TranslateExtIf(comm(send1(C,U)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintChanIntEqif(C,U,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q) || |)
                                    Put("}")
                                    CloseBrackets
                                     Put(";")
                                     Nl
                                     Nl ||  |)


/*-----------------------------------------------------------------TRANSLATE EXTERNAL OTHERWISE-----------------------------------*/

'action' TranslateExtOth(E: EXT_ACT, P:PROCESS, B:BOOLEAN_INT_COMP,Q:PROCESS)
        'rule' TranslateExtOth(comm(bracket(C,V)),P,cons(Cons),Q):
                                    Nl   InsertVarIf(V)
                                    InsertatCif(C,cons(Cons))
                                    PrintVarEq(V,C)
                                    Put("}")
                                    Nl  Put("}")
                                     Nl CloseBrackets
                                     Put(";")
                                     Nl
                                     Nl
        'rule' TranslateExtOth(comm(angle(C,V)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintVarChanEqif(C,V,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
```

```
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q)|| |)
                                    Put("}")
                                     Nl
                                    CloseBrackets
                                     Put("}")
                                     Put(";")
                                     Nl
                                     Nl  || |)
        'rule' TranslateExtOth(comm(send(C,U)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintChanConstantEqif(C,U,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q) || |)
                                    Put("}")
                                    CloseBrackets
                                     Put(";")
                                     Nl
                                     Nl ||  |)
        'rule' TranslateExtOth(comm(send1(C,U)),P,cons(Cons),Q):
                                    (|ChecknotVar(C)
                                    checkl->L
                                    checkl<- lst(varpair(C,1),L)
                                    PrintChanIntEqif(C,U,cons(Cons))
                                    PrintAllDefinitionsfromRest(C,P)
                                    Nl (| ContainsOtherInstancesOfDefinitionCh(C,Q)
                                     Put("otherwise ->")
                                    TranslateExtOtherwise(C,Q) || |)
                                    Put("}")
                                    CloseBrackets
                                     Put(";")
                                     Nl
                                     Nl ||  |)


/*--------------------------------TRANSLATE EXT OTHERWISE-------------------------------------------------------------------------*/

'action' TranslateExtOtherwise(C:CHANNEL,Q:PROCESS)
        'rule' TranslateExtOtherwise(C,zero)
        'rule' TranslateExtOtherwise(channel(C),dotext(E,P)):
                                    PrintAllDefinitionsfromRestExt(C,E,P)
                                     PrintAllDefinitionsfromRest(channel(C),P)
        'rule' TranslateExtOtherwise(channel(C),dotint(I,P)):
                                    PrintAllDefinitionsfromRestInt(C,I,P)



/*--------------------------------TRANSLATE INT IF-------------------------------------------------------------------------*/

'action' TranslateIntIf(C: INT_COMP, P:PROCESS, B: BOOLEAN_INT_COMP,Q:PROCESS)
        'rule' TranslateIntIf(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P,cons(Cons),Q):
                                    (|service->M
                                    ifzero(M)
                                    ifc->L1
                                    ne(L1,nil)
                                    service<-1
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl explorestackif
                                    Nl  Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put(" -> if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
```

```
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                     Put(X1)
                                     Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Nl PrintRestServiceCalls(C,P)
                                    Nl Put("}")
                                    (|ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q)|||)
                                    Put("}")
                                    CloseBrackets Put(";")
                                    TranslateGenCons(genincomp(C),P,cons(Cons),Q)
                                    || service->M
                                    ifzero(M)
                                    service<-1
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl
                                    Nl  Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put(" -> if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                     Put(X1)
                                     Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Nl PrintRestServiceCalls(C,P)
                                    Nl Put("}")
                                    (| ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q)|| |)
                                    Put("}")
                                    CloseBrackets Put(";")
                                    TranslateGenCons(genincomp(C),P,cons(Cons),Q)
                                    || TranslateGenCons(genincomp(C),P,cons(Cons),Q)|)
'rule' TranslateIntIf(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P,cons(Cons),Q):
                                    (|service->M
                                     ifc->L1
                                     ne(L1,nil)
                                     ifzero(M)
                                     service<-1
                                     Nl Put("cServiceInvoke = initially null then if {")
                                     Nl explorestackif
                                     Nl  Put("[]")
                                     Put("@T(")
                                     Printcons(Cons)
                                     Put(")")
                                     Put(" -> if{")
                                     Nl Put("[]")
                                     id_to_string(X->X1)
```

```
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Nl Put("}")
                                    PrintAllServicesFromRest(P)
                                    (|ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q)|| |)
                                    Put("}")
                                    CloseBrackets
                                    Put(";")
                                    TranslateGenCons(genexact(C),P,cons(Cons),Q)
                                    || service->M
                                    ifzero(M)
                                    service<-1
                                    ifc->L1
                                    ne(L1,nil)
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl  Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put(" -> if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Nl Put("}")
                                    PrintAllServicesFromRest(P)
                                    (|ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q)|| |)
                                    Put("}")
                                    CloseBrackets
                                    Put(";")
                                    TranslateGenCons(genexact(C),P,cons(Cons),Q)
                                    || TranslateGenCons(genexact(C),P,cons(Cons),Q) |)
'rule' TranslateIntIf(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P,cons(Cons),Q):
                                   (|service->M
                                    ifzero(M)
                                    ifc->L1
                                    ne(L1,nil)
                                    service<-1
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl explorestackif
                                    Nl Put("[]")
```

```
                                Put("@T(")
                                Printcons(Cons)
                                Put(")")
                                Put(" -> if{")
                                Nl Put("[]")
                                id_to_string(X->X1)
                                Put(" \"true\" ->")
                                id_to_string(I->I1)
                                id_to_string(J->J1)
                                Put("\"")
                                Put(I1)
                                Put(":")
                                Put(J1)
                                Put("(\"")
                                Put("+")
                                Put("\"")
                                Put(X1)
                                Put("\"")
                                Put("+")
                                Put("\")\"")
                                Put("")
                                Nl PrintRestServiceCalls(C,P)
                                Put("}")
                                Nl  (|ContainsServices(Q)
                                Put("otherwise ->")
                                TranslateOtherwiseInt(Q)|| |)
                                Put("}")
                                CloseBrackets
                                Put(";")
                                TranslateGenCons(genincomp(C),P,cons(Cons),Q)
                                || service->M
                                ifzero(M)
                                service<-1
                                ifc->L1
                                ne(L1,nil)
                                Nl Put("cServiceInvoke = initially null then if {")
                                Nl  Put("[]")
                                Put("@T(")
                                Printcons(Cons)
                                Put(")")
                                Put(" -> if{")
                                Nl Put("[]")
                                id_to_string(X->X1)
                                Put(" \"true\" ->")
                                id_to_string(I->I1)
                                id_to_string(J->J1)
                                Put("\"")
                                Put(I1)
                                Put(":")
                                Put(J1)
                                Put("(\"")
                                Put("+")
                                Put("\"")
                                Put(X1)
                                Put("\"")
                                Put("+")
                                Put("\")\"")
                                Put("")
                                Nl PrintRestServiceCalls(C,P)
                                Put("}")
                                Nl (|ContainsServices(Q)
                                Put("otherwise ->")
                                TranslateOtherwiseInt(Q)|| |)
                                Put("}")
                                CloseBrackets
                                Put(";")
                                TranslateGenCons(genincomp(C),P,cons(Cons),Q)
                                || TranslateGenCons(genincomp(C),P,cons(Cons),Q) |)
'rule' TranslateIntIf(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P,cons(Cons),Q):
                               (|service->M
                                ifzero(M)
                                service<-1
                                Nl Put("cServiceInvoke = initially null then if {")
```

```
                                    Nl explorestackif
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put(" -> if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    Put(" \"true\" ->")
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put("+")
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put("+")
                                    Put("\")\"")
                                    Put("")
                                    Nl  Put("}")
                                    Nl PrintAllServicesFromRest(P)
                                    Nl  (|ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q)|| |)
                                    Put("}")
                                    CloseBrackets
                                    Put(";")
                                    TranslateGenCons(genexact(C),P,cons(Cons),Q)
                                    || service->M
                                    ifzero(M)
                                    service<-1
                                    ifc->L1
                                    ne(L1,nil)
                                    Nl Put("cServiceInvoke = initially null then if {")
                                    Nl explorestackif
                                    Nl Put("[]")
                                    Put("@T(")
                                    Printcons(Cons)
                                    Put(")")
                                    Put(" -> if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    Put(" \"true\" ->")
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put("+")
                                    Put("\"")
                                    Put(X1)
                                    Put("\"")
                                    Put("+")
                                    Put("\")\"")
                                    Put("")
                                    Nl  Put("}")
                                    Nl PrintAllServicesFromRest(P)
                                    Nl  (|ContainsServices(Q)
                                    Put("otherwise ->")
                                    TranslateOtherwiseInt(Q) || |)
                                    Put("}")
                                    CloseBrackets
                                    Put(";")
                                    TranslateGenCons(genexact(C),P,cons(Cons),Q)
                                    || TranslateGenCons(genexact(C),P,cons(Cons),Q) |)
'rule' TranslateIntIf(if_then_else(cons(Cons),P,Q),P1,cons(Cons1),Q1):
                                    ifc->L
```

```
                                    PushConstraintIf(cons(Cons1),L)
                                    TranslateProcessIf(P,Q,cons(Cons))


/*---------------------TRANSLATE OTHERWISE INT-----------------------------------------------------------------------*/

'action' TranslateOtherwiseInt(P: PROCESS)
        'rule' TranslateOtherwiseInt(parproc(P,Q))
        'rule' TranslateOtherwiseInt(dotext(E,P)):
                                    TranslateOtherwiseInt(P)
        'rule' TranslateOtherwiseInt(dotint(C,P)):
                                    TranslateOtherwiseIntRest(C,P)
        'rule' TranslateOtherwiseInt(zero)



/*--------------------TRANSLATE OTHERWISE INT REST-----------------------------------------------------------------*/

'action' TranslateOtherwiseIntRest(C: INT_COMP, P:PROCESS)
        'rule' TranslateOtherwiseIntRest(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P):
                                    Nl Put(" if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                     Put(X1)
                                     Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Put("}")
                                    Nl PrintRestServiceCalls(C,P)
                                    Nl Put("};")
        'rule' TranslateOtherwiseIntRest(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P):
                                    Nl Put(" if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    atCvarlist(varlist(V,L))
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
                                    Put(I1)
                                    Put(":")
                                    Put(J1)
                                    Put("(\"")
                                    Put(" + ")
                                    PrintVarlist(varlist(V,L))
                                    Put("\"")
                                     Put(X1)
                                     Put("\"")
                                    Put(" + ")
                                    Put("\")\"")
                                    Put("")
                                    Put("}")
                                    Nl Put("};")
        'rule' TranslateOtherwiseIntRest(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P):
                                    Put("if{")
                                    Nl Put("[]")
                                    id_to_string(X->X1)
                                    Put(" \"true\" ->")
                                    id_to_string(I->I1)
                                    id_to_string(J->J1)
                                    Put("\"")
```

```
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put("+")
                                        Put("\"")
                                         Put(X1)
                                         Put("\"")
                                        Put("+")
                                        Put("\")\"")
                                        Put("")
                                        Put("}")
                                        Nl PrintRestServiceCalls(C,P)
        'rule' TranslateOtherwiseIntRest(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P):
                                        Put("if{")
                                        Nl Put("[]")
                                        id_to_string(X->X1)
                                        Put(" \"true\" ->")
                                        id_to_string(I->I1)
                                        id_to_string(J->J1)
                                        Put("\"")
                                        Put(I1)
                                        Put(":")
                                        Put(J1)
                                        Put("(\"")
                                        Put("+")
                                        Put("\"")
                                         Put(X1)
                                         Put("\"")
                                        Put("+")
                                        Put("\")\"")
                                        Put("")   Put("}") PrintAllServicesFromRest(P) Put("};") Nl
        'rule' TranslateOtherwiseIntRest(if_then_else(cons(Cons),P,Q),P1):
                                        Put("if{")
                                        Nl Put("[]")
                                        Put("@T(")
                                        Printcons(Cons)
                                        Put(" )->")
                                        TranslateOtherwiseInt(P)
                                        Nl Put("otherwise ->")
                                        Nl TranslateOtherwiseInt(Q)
                                        Nl Put("};")


/*----------------------TRANSLATE GENERAL CONSTRAINTS-----------------------------------------------------*/

'action' TranslateGenCons(C:GEN_ACT,P:PROCESS,B:BOOLEAN_INT_COMP,Q:PROCESS)
        'rule' TranslateGenCons(genexact(C),P,cons(Cons),Q):
                                        TranslateExtIf(C,P,cons(Cons),Q)
                                         TranslateProcessIf(P,Q,cons(Cons))
        'rule' TranslateGenCons(genincomp(C),P,cons(Cons),Q):
                                        TranslateIntIf(C,P,cons(Cons),Q)
        'rule' TranslateGenCons(dotgenact(C,D),P,cons(Cons),Q):
                                        TranslateGenCons(C,P,cons(Cons),Q)
                                        TranslateGenCons(D,P,cons(Cons),Q)



/*--------------------TRANSLATE GENERAL CONSTRAINTS OTHERWISE-------------------------------------------*/

'action' TranslateGenConsOth(C:GEN_ACT,P:PROCESS,B:BOOLEAN_INT_COMP,Q:PROCESS)
        'rule' TranslateGenConsOth(genexact(C),P,cons(Cons),Q):
                                        TranslateExtOtherwiseModule(C,Q,cons(Cons))
                                         TranslateProcessIf(P,Q,cons(Cons))
        'rule' TranslateGenConsOth(genincomp(C),P,cons(Cons),Q):
                                        TranslateIntIf(C,P,cons(Cons),Q)
        'rule' TranslateGenConsOth(dotgenact(C,D),P,cons(Cons),Q):
                                        TranslateGenConsOth(C,P,cons(Cons),Q)
                                        TranslateGenConsOth(D,P,cons(Cons),Q)
```

```
/*-----------------------TRANSLATE GENERAL ACTIONS-------------------------------------------------------------*/

'action' TranslateGen(C:GEN_ACT,P:PROCESS)
        'rule' TranslateGen(genexact(C),P):
                                TranslateExt(C,P)
                                 TranslateProcess(P)
        'rule' TranslateGen(genincomp(C),P):
                                TranslateInt(C,P)
        'rule' TranslateGen(dotgenact(C,D),P):
                                TranslateGen(C,P)
                                TranslateGen(D,P)



/*-------------------------PRINTING CONSTRAINTS-----------------------------------------------------------------*/

'action' Printcons(Cons: CONSTRAINT)
        'rule' Printcons(gtc(U,V)):
                                PrintUntypedVariable(U)
                                Put(">")
                                Printnumber(V)
        'rule' Printcons(ltc(U,V)):
                                PrintUntypedVariable(U)
                                Put("<")
                                Printnumber(V)
        'rule' Printcons(geq(U,V)):
                                PrintUntypedVariable(U)
                                Put(">=")
                                Printnumber(V)
        'rule' Printcons(leq(U,V)):
                                PrintUntypedVariable(U)
                                Put("<=")
                                Printnumber(V)
        'rule' Printcons(eqeqcons(U,V)):
                                PrintUntypedVariable(U)
                                Put("==")
                                Printnumber(V)
        'rule' Printcons(gtcv(U,V)):
                                PrintUntypedVariable(U)
                                Put(">")
                                PrintUntypedVariable(V)
        'rule' Printcons(ltcv(U,V)):
                                PrintUntypedVariable(U)
                                Put("<")
                                PrintUntypedVariable(V)
        'rule' Printcons(geqv(U,V)):
                                PrintUntypedVariable(U)
                                Put(">=")
                                PrintUntypedVariable(V)
        'rule' Printcons(leqv(U,V)):
                                PrintUntypedVariable(U)
                                Put("<=")
                                PrintUntypedVariable(V)
        'rule' Printcons(eqeq(U,V)):
                                PrintUntypedVariable(U)
                                Put("==")
                                PrintUntypedVariable(V)
        'rule' Printcons(ltcvc(U,V,C)):
                                PrintUntypedVariable(U)
                                Put("<")
                                PrintUntypedVariable(V)
                                Put("+")
                                Printnumber(C)
        'rule' Printcons(leqvc(U,V,C)):
                                PrintUntypedVariable(U)
                                Put("<=")
                                PrintUntypedVariable(V)
                                Put("+")
                                Printnumber(C)
        'rule' Printcons(geqvc(U,V,C)):
                                PrintUntypedVariable(U)
                                Put(">=")
                                PrintUntypedVariable(V)
```

```
                                    Put("+")
                                    Printnumber(C)
        'rule' Printcons(eqeqc(U,V,C)):
                                    PrintUntypedVariable(U)
                                    Put("==")
                                    PrintUntypedVariable(V)
                                    Put("+")
                                    Printnumber(C)
        'rule' Printcons(and(U,V)):
                                    Printcons(U)
                                    Put("&&")
                                    Printcons(V)


/*-----------------------PRINTING NUMBERS-----------------------------------------------------------------------*/

'action' Printnumber(C: INT)
        'rule' Printnumber(C):
                                    PutI(C)



/*-----------------------PRINT UNTYPED VARIABLE-----------------------------------------------------------------*/

'action' PrintUntypedVariable(U: UNTYPED_VAR)
        'rule' PrintUntypedVariable(untyped_var(U)):
                                     id_to_string(U->U1)
                                     Put(U1)
        'rule' PrintUntypedVariable(untyped_string(U)):
                                    id_to_string(U->U1)
                                     Put("\"")
                                     Put(U1)
                                     Put("\"")
        'rule' PrintUntypedVariable(empty_string):
                                    Put("\"")
                                     Put("\"")


/*-----------------------PRINTING VARIABLE LIST-----------------------------------------------------------------*/

'action' PrintVarlist(L:VARLIST)
        'rule' PrintVarlist(varlist(V,L)):
                                    PrintvariableNoType(V)
                                    Put(" + ")
                                    Put("\",\"")
                                    (lne(L,nil)
                                     Put(" + ")
                                    PrintVarlist(L)
                                     || PrintVarlist(L)|)
        'rule' PrintVarlist(nil):
                                    Put("+")


/*-----------------------PRINTING VARIABLE----------------------------------------------------------------------*/

'action' Printvariable(V: VAR)
        'rule' Printvariable(var(type(T),X)):
                                    id_to_string(T->T0)
                                    Put(T0)
                                    Put(" ")
                                    id_to_string(X->X1)
                                    Put(X1)


/*-----------------PRINT VARIABLE WITH NO TYPE-----------------------------------------------------------------*/

'action' PrintvariableNoType(V: VAR)
        'rule' PrintvariableNoType(var(type(T),X)):
                                    Put(" ")
                                    id_to_string(X->X1)
                                    Put(X1)
```

```
/*-------------------------PRINTING CHANNEL-------------------------------------------------------------------------*/

'action' PrintChannel(C: CHANNEL)
        'rule' PrintChannel(channel(var(type(T),X))):
                                Nl id_to_string(T->T0)
                                Put(T0)
                                Put(" ")
                                id_to_string(X->X1)
                                Put(X1)
                                Put(";") Nl

/*-----------------------------CONDITIONS-----------------------------------------------------------------------------*/


'condition' IsStack(I:INT)
        'rule' IsStack(I):
                                 stack->J
                                 eq(I,J)


'condition' ifzero(I:INT)
        'rule' ifzero(I):
                                 eq(I,0)


'condition' checkdeclaredvariable(C:CHANNEL)
        'rule' checkdeclaredvariable(channel(var(T,X))):
                                declaredvariable -> L
                                eq(L,nil)
        'rule'    checkdeclaredvariable(channel(var(T,X))):
                                declaredvariable->chanlist(U,V)
                                notchannelsearch(channel(var(T,X)),U)
                                notchannelsearchlist(channel(var(T,X)),V)



'condition' ChecknotVar(V:CHANNEL)
        'rule'    ChecknotVar(channel(var(T,X))):
                                checkl->L
                                eq(L,nil)
        'rule'    ChecknotVar(channel(var(T,X))):
                                checkl->lst(U,V)
                                notsearch(var(T,X),U)
                                notsearchlist(var(T,X),V)



'condition' notsearch(V:VAR, U:VARPAIR)
        'rule'  notsearch(var(T,X),varpair(channel(var(T1,X1)),N)):
                                (| ne(T,T1)
                                || ne(X,X1)|)



'condition' notchannelsearch(C:CHANNEL, U:CHANNEL)
        'rule'  notchannelsearch(channel(var(T,X)),channel(var(T1,X1))):
                                (| ne(T,T1)
                                || ne(X,X1)|)



'condition' notsearchlist(V:VAR, C:CHECKLIST)
        'rule'  notsearchlist(var(T,X),nil)
        'rule'  notsearchlist(var(T,X),lst(varpair(U,N), W)):
                                notsearch(var(T,X),varpair(U,N))
                                notsearchlist(var(T,X),W)



'condition' notchannelsearchlist(C:CHANNEL, D:CHANLIST)
        'rule'  notchannelsearchlist(channel(var(T,X)),nil)
```

```
        'rule'  notchannelsearchlist(channel(var(T,X)),chanlist(C, W)):
                                notchannelsearch(channel(var(T,X)),C)
                                notchannelsearchlist(channel(var(T,X)),W)


'condition' ContainsServices(P: PROCESS)
        'rule' ContainsServices(dotext(E,P)):
                                ContainsServices(P)
        'rule' ContainsServices(dotint(C,P)):
                                (|ContainsServicesInt(C,P)|| ContainsServices(P)|)


'condition'  ContainsServicesInt(C: INT_COMP, P:PROCESS)
        'rule' ContainsServicesInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P)
        'rule'  ContainsServicesInt(let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P)
        'rule'  ContainsServicesInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P)
        'rule'  ContainsServicesInt(let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P)
        'rule'  ContainsServicesInt(if_then_else(cons(Cons),P,Q),P1):
                                (|  ContainsServices(P)
                                ||   ContainsServices(Q)|)


'condition' ContainsOtherInstancesOfDefinition(V:VAR,P: PROCESS)
        'rule' ContainsOtherInstancesOfDefinition(V,dotext(E,P)):
                                (|ContainsDefinitionExt(V,E) ||
                                ContainsOtherInstancesOfDefinition(V,P)|)
        'rule' ContainsOtherInstancesOfDefinition(V,dotint(C,P)):
                                ContainsDefinitionInt(V,C,P)


'condition' ContainsDefinitionExt(V:VAR,E: EXT_ACT)
        'rule' ContainsDefinitionExt(var(T,X),comm(bracket(C,var(T1,X1)))):
                                eq(T,T1)
                                 eq(X,X1)
        'rule' ContainsDefinitionExt(var(T,X),comm(angle(channel(var(T1,X1)),V))):
                                eq(T,T1)
                                 eq(X,X1)
        'rule' ContainsDefinitionExt(var(T,X),comm(send(channel(var(T1,X1)),U))):
                                eq(T,T1)
                                 eq(X,X1)
        'rule' ContainsDefinitionExt(var(T,X),comm(send1(channel(var(T1,X1)),U))):
                                eq(T,T1)
                                 eq(X,X1)


'condition' ContainsDefinitionInt(V: VAR, C: INT_COMP, P:PROCESS)
        'rule' ContainsDefinitionInt(V1,let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genincomp(C)),P):
                                 (| ContainsDefinitionInt(V1,C,P)
                                 || ContainsOtherInstancesOfDefinition(V1,P)|)
        'rule' ContainsDefinitionInt(V1,let_instantiate(var(type(T),X),method(T1,I,J,varlist(V,L)),genexact(C)),P):
                                (| ContainsDefinitionExt(V1,C)
                                 || ContainsOtherInstancesOfDefinition(V1,P)|)
        'rule' ContainsDefinitionInt(V1,let_instantiate(var(type(T),X),method(T1,I,J,nil),genincomp(C)),P):
                                 ContainsOtherInstancesOfDefinition(V1,P)
        'rule' ContainsDefinitionInt(V1,let_instantiate(var(type(T),X),method(T1,I,J,nil),genexact(C)),P):
                                (| ContainsDefinitionExt(V1,C)
                                 || ContainsOtherInstancesOfDefinition(V1,P)|)
        'rule' ContainsDefinitionInt(V1,if_then_else(cons(Cons),P,Q),P1):
                                (|  ContainsOtherInstancesOfDefinition(V1,P)
                                 ||  ContainsOtherInstancesOfDefinition(V1,Q)|)


'condition' ContainsPrintableType(V: VARLIST)
        'rule' ContainsPrintableType(varlist(var(type(T),X),L)):
                                 (| id_to_string(T->T1)
                                 ne(T1,"aldef")
                                 || ContainsPrintableType(L) |)


'condition'  ContainsOtherInstancesOfDefinitionCh(C:CHANNEL,Q: PROCESS)
         'rule' ContainsOtherInstancesOfDefinitionCh(channel(V),Q):
```

ContainsOtherInstancesOfDefinition(V,Q)

'action' id_to_string(IDENT->STRING)

'type' PRN
module