September 6, 2000

(Author's Name)

University of Maryland

A.S. Subrahmanian

HETEROGENEOUS AGENT SYSTEMS
A piece of software code may be described via three components:

1. \( S \): the set of all data types managed by \( S \).
2. \( F \): a set of predefined functions on these types.
3. \( C \): a set of type composition operations.

Software Code Abstractions
Type composition operations

1. A partial function on types.
2. Takes as input types $T_1, \ldots, T_n$.
3. Output is a type $C(T_1, \ldots, T_n)$. 
We will assume $S_L = S_L$, i.e., $S_L$ is closed under type composition:

$$S_L \cup \exists \circ \ne \ne \circ f \circ = S_L$$

$$(S_L \circ S_L) \circ f \circ = S_L$$

$$S_L \circ f \circ = 0_L$$
Location is an enumerated type of city names.

Stock has the schema (amount/Integer, part-id/String),

dest/Location, pickup-st/date, method/String, src/Location,

OrderLog has the schema (client/String, client/Date, orderLog/Stock,

Where:

\{OrderLog, Location, part-id/String, \} \cup \{OrderLog, Stock, Location, part-id/String, \} f \cup \{OrderLog, Location, part-id/String, \}
Updates the inventory of the Supplier:

\[
\text{updateStock} \ (\text{Amount}/\text{Integer}, \ \text{Part-id}/\text{String})
\]

- By the shipping vendor.

Returns either amount available or amount not available:

\[
\text{monitorStock} \ (\text{Amount}/\text{Integer}, \ \text{Part-id}/\text{String})
\]

Suppliers' agents: Functions
Supplier agents: Type Composition Operators

1. Projection
2. Cartesian Product

$T_s$ consists not only of all the above data types, but also of
1. sub-records of the above data types such as a relation having the
   schema (amount/Integer, dest/Location) derived from the OrderLog
   relation,
2. cartesian products of types, and
3. mixes involving the above two operations.
State of an Agent

Agent has a set of data structures.

State of the agent, denoted $S(t)$ at time $t$, consists of all objects in the agent's data structures.

Each agent is assumed to have some specialized message data structures and message management functions which we will discuss later on in the course.

An agent may change its state by taking an action—either triggered internally, or by processing a message received from another agent.

An agent may change its state by taking an action—either triggered internally, or by processing a message received from another agent.

Each agent is assumed to have some specialized message data structures and message management functions which we will discuss later on in the course.
Given a type variable \( \tau \), we assume that there is a set \( \mathcal{T} \) of variables.

For any path variable \( \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z} \), we require that \( \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z} \) is a variable of type \( \mathcal{Z} \).

Similarly, \( \mathcal{X} \cdot \mathcal{Y} \cdot \mathcal{Z} \) is a variable of type \( \mathcal{Z} \), and so on.

Given any type \( \tau \in \mathcal{T} \), we will assume that there is a set \( \mathcal{T} \) of variables.
$X$ is a root variable over $\text{OrderLog}$. 

$X$ is a path variable over strings with root $X$. 

$X$ is a root variable over $\text{OrderLog}$. 

$X$ is a path variable over locations. 

$X$ is a root variable over $\text{OrderLog}$. 

$X$ is a path variable over the integers $\{1, \ldots, 12\}$. 

Example
An assignment of objects to variables is a set of equations of the form

\[ V_i = o_i \]

where the \( V_i \)’s are variables (root or path) and the \( o_i \)’s are objects. Such an assignment is legal if the types of objects and corresponding variables match.

Example:

\[ \{ \text{client} = \text{john}, \text{date.month} = 5, \text{date.year} = 1998 \} \]

An assignment of objects to variables is a set of equations of the form

\[ \text{assignment} \]
Code Calls

Each API function has a signature, specifying the types of inputs it takes, and the types of outputs it returns.

Not all API functions may be used by an agent.

Invoke an API function supported by a software package.

Syntactic mechanism that allows a process (which may be an agent) to
Every code call is assumed to return a SET.

A code call is ground if all the \( d_i \) are objects.

Then \( S:f(d_1, \ldots, d_n) \) is a code call.

If \( d_i \) respects the type requirements of the \( i \)th argument of \( f \) and \( d_1, \ldots, d_n \) are objects or variables such that each \( d_i \) respects the type

Suppose \( f \in F \subseteq S \) is a predefined function with \( n \) arguments, and
Example Code Calls

In contrast, the preceding code call is ground, and is executable.

This code call says that we should check whether 3 pieces of part 008 are available or not. The result of this call is either the singleton set $\{\text{amount available}\}$, or the set $\{\text{amount not available}\}$ are available or not. The result of this call is either the singleton set $\{\text{amount available}\}$, or the set $\{\text{amount not available}\}$ are available or not.

This code call says that we should create a pickup schedule for shipping 3 pieces of part 008 from location X to Paris by truck. Notice that until a value is specified for X, this code call cannot be executed. In contrast, the preceding code call is ground, and is executable.
A code call atom is ground if no variable symbols occur anywhere in it.

- notin(X,cc)
- in(X,cc)

If cc is a code call, and X is either a variable symbol or an object of

Code Call Atoms
Code Call Atom Evaluation

In database terminology, $X$ is a cursor on the result of executing the code call.

Similarly for `not in`.

$\text{Code Call}$(X, cc) succeeds if $X$ can be set to a pointer to one of the objects in the set of objects returned by executing the code call.

$\text{Code Call}$(X; cc) succeeds if $X$ can be set to a pointer to one of the objects in the set of objects returned by executing the code call.
Example Code Call Atoms

supplier : monitorstock(3, part-008).

This code call succeeds just in case the Supplier has 3 units of part 008 on stock.

In effect, X is a variable that can be bound to the status string returned by executing the code call shown here. In the result of executing the code call shown here, there exists an X in the result of in(X, supplier: monitorstock(3, part-008)).
Intuitively a "conjunction" (logical AND) of code call atoms and comparison atoms are expressions such as ",=", ",\geq\", ",\leq\", ",<\", ",\gg\", ",\ll\", etc.

Comparison atoms are expressions such as ",=\", ",\geq\", ",\leq\", ",<\", etc.

Code Call Conditions
Every code call atom is a code call condition.

If $s$ and $t$ are either variables or objects, then $s = t$ is a code call condition.

If $s$ and $t$ are either variables or objects, or are variables over the integers/real valued objects, or are variables over the integers/real values, then $s \leq t$, and $s > t$, $s < t$, $s > t$ are code call conditions.

If $X_1$ and $X_2$ are code call conditions, then $X_1 \land X_2$ is a code call condition.

Copyright, V.S. Subrahmanian, et al.
Example Code Call Conditions
Example Code Call Conditions

\[ A > \lambda \]
\[ \gamma \text{notIn} \ (\text{amountAvailable} \land \text{supplier} : \text{monitorStock} (A + 1, \text{part} 009)) \]
\[ \gamma \text{in} \ (\text{amountAvailable} \land \text{supplier} : \text{monitorStock} (A, \text{part} 008)) \]
\[ \gamma \text{notIn} \ (\text{amountAvailable} \land \text{supplier} : \text{monitorStock} (A + 1, \text{part} 008)) \]
\[ \gamma \text{in} \ (\text{amountAvailable} \land \text{supplier} : \text{monitorStock} (A, \text{part} 009)) \]
In-class exercise

You have a relational database, rel. Rel has a table called inventory.
In-class exercise

Give a list of API functions that might be supported by REL.
in-class exercise

Give a similar specification of a simple calculator program (e.g., XCALC).
In-class exercise

Write code calls for the following tasks using the functions given on the previous 2 slides.

- Find all books by an author whose last name is Cook.
- Find all books whose Qty is less than 10.
- Find all books whose restock quantity is more than 5.
In-class exercise

Write code call conditions.

Find all books whose restock quantity is 3 or less below the Qty?

Find all books by John Keats whose restock quantity is 3 or less below the Qty?

Find all books by John Keats published by ABC Books whose restock quantity is 3 or less below the Qty?
Can we reorder the code call atoms occurring in it in a way such that we can evaluate these atoms left to right, assuming that root variables are incrementally bound to objects? A code call \( S : f(d_1, \ldots, d_n) \) is safe if and only if each \( d_i \) is ground.

\[ \text{Safety} \]
A code call condition is safe if and only if there exists a permutation of the form in which the root of each variable occurring in the code call atom of the form in is from the root of at least one of the roots of the code call atom of the form in, such that:

\[ (\forall \lambda X) \text{ root is from } (\forall X) \text{ root belongs to } (\forall \lambda) \subseteq (X) \text{ root } \]

1. If \( s \) is neither a constant nor a variable \( X \) such that \( \{ (\forall X) \text{ root occurs in } \lambda \text{ at least once} \} \subseteq (\forall \lambda) \text{ root } \) \( \forall \varphi = (\forall \lambda \text{ root belongs to } (X) \text{ root } \)

2. If \( s \) is a root variable, then at least one of \( s \)'s \( \varphi \) is a constant or a variable \( X \) such that \( (\forall X) \text{ root } \)

3. If \( s \) is a comparison or \( \varphi \), then \( (\forall X) \text{ root } \)

The following holds:

- There exists a permutation of the form in such that for every \( i \) such that \( \forall X \text{ root } \), there is safe if and only if there is a code call condition in.

\[ \text{Safety} \]
Example

Consider our old three code call conditions. Which ones are safe?
What is the relationship between the problem of checking safety, and the problem of checking safety modulo a set of variables? Suppose \( X \) is a code call condition, and let \( \theta \) be any set of root variables. Then, \( X \) is said to be safe modulo \( X \) if and only if for an arbitrary (assignment of objects to the variables in \( X \)), it is the case that is safe.
Algorithm safe-cc

1. input is a code call condition
2. if \( \mu = m \) then return unsafe
3. else
4. select all \( i \) from \( L \) such that \( i \) is safe modulo \( X \)
5. if \( m = 0 \) then return unsafe
6. otherwise, the output is unsafe
7. remove all \( i \) from \( L \)
8. \( X = X \cap \{ (\lambda) \} \)
9. output is a proper reordering
10. end

\( \text{safe-cc}(\lambda): \text{code call condition}; X: \text{set of root variables} \)
Theorem

Suppose \( \theta \), \( \chi \) is a safe code call condition which can be evaluated left-to-right, \( \chi \) is safe modulo a set of root variables \( X \) such that \( \text{safe-cc}(\chi, X) \) returns a reordering of \( X \). Moreover, for any assignment to the variables in \( X \), \( \chi \) is safe if and only if \( \text{safe-cc}(\chi, X) \) returns a reordering of \( X \). Then, \( \chi \) is a code call condition.
Consider \( x_1 \) and \( x_2 \), where

\[
\chi = \chi_1 \chi_2
\]

On the first iteration \( m = 2 \), and

\[
\chi \equiv \chi_1 \chi_2 \equiv \chi
\]

\( \chi \) is available.

\( \chi \) is in \( \chi \), supplier: muniStock(3), part-008

2. We add \( x_1 \) and \( x_2 \) to \( \chi \), remove them from \( L \), and update \( X \).

3. Now, \( L \) is empty, and the algorithm correctly returns \( X \).
What happens when we run the algorithm?

\[ \Lambda > \in \chi \]

Example

Consider \( \chi \times \chi \times \chi \times \chi \times \chi \times \chi \), where
Clearly, by construction of $X$, we have that $X$ is a safe code call.

Thus, if the call of safe-ccc$(X, X)$ returns, then $X$ is safe modulo $\mathcal{X}$.

Condition modulo $\mathcal{X}$ (a suitable permutation for $X$, the reordering of the constituents of $X$, as a suitable permutation for $X$ is given by the reordering of the constituents of $X$).

Proof of Theorem (part)
Suppose \( \chi \) is safe modulo \( X \). Then there exists an \( i \) such that \( \chi_i \) is safe modulo \( X \). Obviously, \( \chi \) is safe, and for any assignment to the variables in \( X \), \( \chi \) returns a reordered code call condition \( \chi(X,\cdot) \). Hence the computation succeeds for \( I \) which are all root variables, \( \chi \) is safe, and \( X \) is reordered code call condition.

Suppose \( X \) is safe modulo \( X \). Then there exists an \( i \) such that \( \chi_i \) is safe modulo \( X \).

Employing an inductive argument, we obtain that the continuation of the computation succeeds for \( I \) which are all root variables. Hence the computation succeeds for \( I \) which are all root variables, \( \chi \) is safe, and for any assignment to the variables in \( X \), \( \chi \) can be readily evaluated left-to-right. (which are all root variables), the code call condition \( \theta, \chi \) can be readily evaluated left-to-right.
Complexity of Algorithm

Quadratic in $n$.

Why? Because the number of iterations is bounded by the number $n$ of constituents $\chi_i$ of $\chi$, and the body of the while loop can be executed in linear time.
Suppose \( X = 0 \) is a code call condition involving the variables \( X \), \( Y \), \( Z \), and \( W \). Suppose some software code:

\[
\begin{align*}
\{X, Y, Z, W\} & \in \text{dep} \\
\text{Suppose } X \text{ is a code call condition involving the variables } X, Y, Z, W.
\end{align*}
\]
Example

Suppose there are $n$ units of part-008 on stock and $m$ units of part-009.

$$\text{suppliers : monitorstock(A, part 008)}, \text{suppliers : monitorstock(A, part 009)}$$

If $n > m$, assign $n$ to $A$ and $m$ to $\Lambda$. If $n < m$, assign $m$ to $A$ and $n$ to $\Lambda$. If $n = m$, assign $n$ to $A$ and $m$ to $\Lambda$. If $n = m = 0$, assign $n$ to $A$ and $m$ to $\Lambda$. If $n = m = 0$, assign $n$ to $A$ and $m$ to $\Lambda$.
An integrity constraint $IC$ is an expression of the form $\exists \phi$ where

$\phi$ is an atomic code call condition such that every root variable in $\chi$ occurs in $\phi$.

$\chi \Leftarrow \phi$
This IC says that at least three units of part-008 must always be available.

\[
\text{IC} \subseteq 800\text{ part} = p \land \text{in quantity available, supplier: monitorstock (3, P)}.
\]
Example
This non-IC says that the amount of part available is at least as high as the sum of the number of part available and the number of part available.
In-class exercise:

Write an IC (for the inventory relation) saying that: Qty must always be a least 1.2 times Rstk.
In-class exercise

Write an IC (for the inventory relation) saying that: Qty must always be at least 1.3 times Rstk for all books by Tom Clancy.
In-class exercise:

Write an IC (for the Inventory relation) saying that: Qty must always be at least 1.25 times Rest stock for all books published by XYZ Books.
A state $s$ satisfies an integrity constraint $IC$ of the form $\phi$ if for every legal assignment of objects from $O$ to the variables in $IC$, either $\phi$ is false or $\chi$ is true. The variables in $IC$, either $\phi$ is false or $\chi$ is true.

\[ \chi \leftarrow \phi \]

$S$ satisfies a set of integrity constraints if it satisfies each integrity constraint in the set.

Integrity Constraint Satisfaction
\textbf{The is Construct}