HETEROGENEOUS AGENT SYSTEMS

V.S.Subrahmanian
University of Maryland
:
(with P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, R. Ross)

September 6, 2000

Software Code Abstractions

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

1. $\mathcal{T}_{\mathcal{S}}$: the set of all data types managed by $\mathcal{S}$,
2. $\mathcal{F}_{\mathcal{S}}$: a set of predefined functions on these types,
3. $\mathcal{C}_{\mathcal{S}}$: a set of type composition operations.

Type composition operations

1. A partial function c on types.
2. Takes as input types $\tau_1, \ldots, \tau_n$;
3. Output is a type $c(\tau_1, \ldots, \tau_n)$.

$\boxed{\mathcal{T}_{\mathcal{S}}^{\star}}$

$$\begin{aligned}
\mathcal{T}_{\mathcal{S}}^{0} &=_{def} \mathcal{T}_{\mathcal{S}}, \\
\mathcal{T}_{\mathcal{S}}^{i+1} &=_{def} \mathcal{C}_{\mathcal{S}}(\mathcal{T}_{\mathcal{S}}^{i}), \\
\mathcal{T}_{\mathcal{S}}^{\star} &=_{def} \cup_{i \in \mathbf{N}} \mathcal{T}_{\mathcal{S}}^{i}.
\end{aligned}$$

We will assume $\mathcal{T}_{\mathcal{S}} = \mathcal{T}_{\mathcal{S}}^{\star}$, i.e. $\mathcal{T}_{\mathcal{S}}$ is closed under type composition.

Supplier agents: Types

$\mathcal{T}_{\mathcal{S}} =_{def} \{$Integer, Location, String, Date, OrderLog, Stock$\}$

where:

- OrderLog has the schema (client/string, amount/Integer, part_id/String, method/String, src/Location, dest/Location,pickup_st/date, pickup_et/date),
- Stock has the schema (amount/Integer, part_id/String)
- Location is an enumerated type of city names.

Supplier agents: Functions

- *monitorStock*(*Amount/integer*, *Part_id/string*).
  Returns either `amount_available` or `amount_not_available`.
- *shipFreight*(*Amount/Integer*, *Part_id/String*, *method/String*, *Src/Location*, *Dest/Location*).
  Updates the order log and logs information about the order, together with information on (i) the earliest time the order will be ready for shipping, and (ii) the latest time by which the order must be picked up by the shipping vendor.
- *updateStock*(*Amount/Integer*, *Part_id/String*).
  Updates the inventory of the Supplier.

Supplier agents: Type Composition Operators

1. Projection
2. Cartesian Product

$\mathcal{T}_{\mathcal{S}}^{\star}$ 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 $\mathcal{O}_{\mathcal{S}}(t)$ — at time t, consists of all objects in the agent's data structures.
- 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.

Variables

- Given any type $\tau \in \mathcal{T}_{\mathcal{S}}$, we will assume that there is a set root(τ) of "root" variable symbols ranging over τ.
- Suppose τ is a complex record type having fields $\mathrm{f_1}, \ldots, \mathrm{f_n}$. Then, for every variable of type τ, we require that $X.f_i$ be a variable of type τ_i where τ_i is the type of field $\mathrm{f_i}$.
- Similarly, if $\mathrm{f_i}$ itself has a sub-field g of type γ, then $X.f_i.g$ is a variable of type γ, and so on.
- The variables, $X.f_i$, $X.f_i.g$, etc. are called *path variables*.
- For any path variable Y of the form $X.path$, where X is a root variable, we refer to X as the root of Y, denoted by root(Y).

Example

- X is a root variable over OrderLog.
- $X.client$ is a path variable over strings with root X.
- $X.Location$ is a path variable over locations.
- $X.date.month$ is a path variable over the integers $\{1 \ldots 12\}$.

Variable Assignment

- An *assignment of objects to variables* is a set of equations of the form $V_1 = o_1, \ldots, V_k = o_k$ 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.
- EX: $\{X.client =$"john" $\}$
- EX: $\{X.client =$"john", $X.date.month = 5, X.date.year = 1998\}$.

Code Calls

- Syntactic mechanism that allows a process (which may be an agent) to invoke an API function supported by a software package.
- Not all API functions may be used by an agent.
- Each API function has a *signature*, specifying the types of inputs it takes, and the types of outputs it returns.

Code Calls

- Suppose $f \in \mathcal{F}_{\mathcal{S}}$ is a predefined function with n arguments, and $d_1, \ldots, d_n$ are objects or variables such that each d_i respects the type requirements of the i'th argument of f.
- Then $\mathcal{S}$: $\mathrm{f(d_1, \ldots, d_n)}$ is a code call.
- A code call is *ground* if all the $\mathrm{d_i}$'s are objects.
- Every code call is assumed to return a SET.

Example Code Calls

1. supplier : monitorStock(3, part_008).
   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 $\{ amount\_available \}$, or the set $\{ amount\_not\_available \}$.
2. supplier : shipFreight(3, part_008, truck, X, paris).
   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.

Code Call Atoms

- If *cc* is a code call, and X is either a variable symbol, or an object of the output type of *cc*, then
  - in(X,cc),
  - notin(X,cc),

  are called *code call atoms*.
- A code call atom is *ground* if no variable symbols occur anywhere in it.

Code Call Atom Evaluation

- Code call atoms, when evaluated, return boolean values.
- Code call atom in(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.
- In database terminology, X is a cursor on the result of executing the code call.
- Similarly for **notin**.

Example Code Call Atoms

1. in(amount_available,supplier:monitorStock(3, part_008)).
   This code call succeeds just in case the Supplier has 3 units of *part_008* on stock.
2. in(X,supplier:monitorStock(3, part_008)).
   This code call succeeds just in case there exists an X in the result of executing the code call shown here. In effect, X is a variable that can be bound to the status string returned by executing the code call supplier : monitorStock(3, part_008).

Code Call Conditions

- Intuitively a "conjunction" (logical AND) of code call atoms and "comparison" atoms.
- 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 integers/real valued objects, or are variables over the integers/reals, then $s < t$, $s > t$, $s \leq t$, and $s \geq t$ are code call conditions.
- If χ_1 and χ_2 are code call conditions, then $\chi_1 \,\&\, \chi_2$ is a code call condition.

Example Code Call Conditions

$$\chi^{(1)}: \quad \text{in}(\mathit{amount\_available}, \text{supplier} : \text{monitorStock}(3, \text{part\_008})).$$

$$\chi^{(2)}: \quad \text{in}(X, \text{supplier} : \text{monitorStock}(3, \text{part\_008})) \,\&\, X = \mathit{amount\_available}.$$

Example Code Call Conditions

$$
\begin{array}{lll}
\chi^{(3)}: & \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(U, \text{part\_008})) & \& \\
& \text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(U+1, \text{part\_008})) & \& \\
& \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(V, \text{part\_009})) & \& \\
& \text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(V+1, \text{part\_009})) & \& \; U < V.
\end{array}
$$

In-class exercise

You have a relational database, rel. Rel has a table called **`inventory`**.

Lname	Fname	Title	Qty	Rstk	Sup	Buy	Sell
John	Keats	Odes	15	5	ABC Books	20	38
Leon	Uris	Topaz	10	3	XYZ Books	6	11
Robin	Cook	Virus	20	6	A1 Books	4	14
...	...	...	...	...	...	...	...

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

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?

Safety

- Intuition: 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 $\mathcal{S} : \mathrm{f}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n})$ is *safe* if and only if each d_i is ground.

Safety

- A code call condition $\chi_1 \,\&\, \ldots \,\&\, \chi_n$, $n \geq 1$, is *safe* if and only if there exists a permutation π of $\chi_1, \ldots, \chi_n$ such that for every $i = 1, \ldots, n$ the following holds:

  1. If $\chi_{\pi(i)}$ is a comparison $s_1\ op\ s_2$, then

     1.1 at least one of s_1, s_2 is a constant or a variable X such that root(X) belongs to $RV_\pi(i) =_{def} \{\text{root}(Y) \mid \exists j < i \text{ s.t. } Y \text{ occurs in } \chi_{\pi(j)}\}$;

     1.2 if s_i is neither a constant nor a variable X such that root$(X) \in RV_\pi(i)$, then s_i is a root variable.

  2. If $\chi_{\pi(i)}$ is a code call atom of the form in$(X_{\pi(i)}, cc_{\pi(i)})$ or notin$(X_{\pi(i)}, cc_{\pi(i)})$, then the root of each variable Y occurring in $\texttt{cc}_{\pi(\texttt{i})}$ belongs to $RV_\pi(i)$, and either $X_{\pi(i)}$ is a root variable, or root$(X_{\pi(i)})$ is from $RV_\pi(i)$.

Example

Consider our old three code call conditions. Which ones are safe?

$$\chi^{(1)}: \quad \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(3, \text{part\_008})).$$

$$\chi^{(2)}: \quad \text{in}(X, \text{supplier} : \text{monitorStock}(3, \text{part\_008})) \,\&\, X = amount\_available.$$

$$\begin{aligned}\chi^{(3)}: \quad & \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(U, \text{part\_008})) && \& \\ & \text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(U+1, \text{part\_008})) && \& \\ & \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(V, \text{part\_009})) && \& \\ & \text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(V+1, \text{part\_009})) && \&\ U < V.\end{aligned}$$

Safety Modulo Variables

- Suppose χ is a code call condition, and let $\mathbf{X}$ be any set of root variables. Then, χ is said to be *safe modulo* $\mathbf{X}$ if and only if for an (arbitrary) assignment θ of objects to the variables in $\mathbf{X}$, it is the case that $\chi\theta$ is safe.
- What is the relationship between the problem of checking safety, and the problem of checking safety modulo a set of variables?

Algorithm safe_ccc

safe_ccc(χ: code call condition; X: set of root variables)

($\star$ input is a code call condition $\chi = \chi_1 \& \cdots \& \chi_n$; output is a proper reordering $\star$)
($\star$ $\chi' = \chi_{\pi(1)} \& \cdots \& \chi_{\pi(n)}$ if χ is safe modulo **X**; otherwise, the output is *unsafe*; $\star$)

1. L:=$\chi_1, \ldots, \chi_n$;
2. χ:=true;
3. **while** L is not empty **do**
4. { select all $\chi_{i_1}, \ldots, \chi_{i_m}$ from L such that χ_{i_j} is safe modulo **X**;
5. **if** $m = 0$ **then return** *unsafe* (*exit*);
6. **else**
7. { χ:=$\chi \& \chi_{i_1} \& \cdots \& \chi_{i_m}$;
8. remove $\chi_{i_1}, \ldots, \chi_{i_m}$ from L;
9. $\mathbf{X} = \mathbf{X} \cup \{\text{root}(\mathbf{Y}) \mid Y$ occurs in some $\chi_{\mathbf{i_1}}, \ldots \chi_{\mathbf{i_m}}\}$;
10. }
11. }
12. **return** χ';

end.

Theorem

Suppose $\chi =_{def} \chi_1 \,\&\, \ldots \,\&\, \chi_n$ is a code call condition. Then, χ is safe modulo a set of root variables $\mathbf{X}$, if and only if **safe_ccc**$(\chi, \mathbf{X})$ returns a reordering χ' of χ. Moreover, for any assignment θ to the variables in $\mathbf{X}$, $\chi'\theta$ is a safe code call condition which can be evaluated left-to-right.

Example

Consider $\chi_1 \,\&\, \chi_2$, where

χ_1 is $\text{in}(X, \text{supplier} : \text{monitorStock}(3, \text{part\_008}))$
χ_2 is $X = amount\_available.$

1. On the first iteration $m = 2$, $\chi_{i_1} = \chi_1$, and $\chi_{i_2} = \chi_2$.
2. We add χ_{i_1} and χ_{i_2} to χ', remove them from L, and update $\mathbf{X}$.
3. Now, L is empty, and the algorithm correctly returns χ'.

Example

Consider $\chi_1 \,\&\, \chi_2 \,\&\, \chi_3 \,\&\, \chi_4 \,\&\, \chi_5$, where

χ_1 is in($amount\_available$, supplier : monitorStock(U, part_008)),
χ_2 is notin($amount\_available$, supplier : monitorStock(U + 1, part_008)),
χ_3 is in($amount\_available$, supplier : monitorStock(V, part_009)),
χ_4 is notin($amount\_available$, supplier : monitorStock(V + 1, part_009)), *and*
χ_5 is $U < V$.

What happens when we run the algorithm?

Proof of Theorem ($\rightarrow$ part)

- Clearly, by construction of χ', we have that χ' is a safe code call condition modulo $\mathbf{X}$ (a suitable permutation for χ' is the identity).
- Thus, if the call of **safe_ccc**$(\chi, \mathbf{X})$ returns χ', then χ is safe modulo $\mathbf{X}$, as a suitable permutation for π is given by the reordering of the constituents of χ realized in χ'.

Proof of Theorem ($\leftarrow$ part)

- Suppose χ is safe modulo $\mathbf{X}$.
- Then there exists an $i \in \{1, \ldots, n\}$ such that χ_i is safe modulo $\mathbf{X}$.
- If, without loss of generality, $\chi_1, \ldots, \chi_k$ are all such χ_i, then it is easily seen that $\chi^{(1)} =_{def} \chi_{k+1} \& \cdots \& \chi_m$ must be safe modulo the variables $\mathbf{X^{(1)}} =_{\mathbf{def}} \mathbf{X} \cup \{\mathrm{root}(\mathbf{Y}) \mid \mathtt{Y}$ occurs in $\chi_1 \& \cdots \& \chi_{\mathtt{k}}\}$.
- Employing an inductive argument, we obtain that the continuation of the computation succeeds for $L =_{def} \chi_{k+1}, \ldots, \chi_n$ modulo $\mathbf{X^{(1)}}$.
- Hence **safe_ccc**$(\chi, \mathbf{X})$ returns a reordered code call condition χ'.
- Obviously, χ' is safe, and for any assignment θ to the variables in $\mathbf{X}$ (which are all root variables), the code call condition $\chi'\theta$ can be readily evaluated left -to-right.

Complexity of Algorithm

- Quadratic in n.
- Why? Becuase the number of iterations is bounded by the number n of constituents χ_i of χ, and the body of the while loop can be executed in linear time.

CAN YOU DO BETTER THAN THIS?

Code Call Solution

- Suppose χ is a code call condition involving the variables $\mathbf{X}=_{\mathbf{def}}\{\mathbf{X_1}, \ldots, \mathbf{X_n}\}$;
- Suppose $\mathcal{S}=_{def}(\mathcal{T}_\mathcal{S}, \mathcal{F}_\mathcal{S}, \mathcal{C}_\mathcal{S})$ is some software code;
- A *solution* of χ w.r.t. $\mathcal{T}_\mathcal{S}$ in a state $\mathcal{O}_\mathcal{S}$ is a legal assignment of objects $o_1, \ldots, o_n$ to the variables $X_1, \ldots, X_n$, written as a compound equation $\mathbf{X}:=\mathbf{o}$, such that the application of the assignment makes χ true in state $\mathcal{O}_\mathcal{S}$.

Example

$$
\begin{array}{ll}
\text{in}(amount\_available, \text{supplier} : \text{monitorStock}(U, \text{part\_008})) & \& \\
\text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(U+1, \text{part\_008})) & \& \\
\text{in}(amount\_available, \text{supplier} : \text{monitorStock}(V, \text{part\_009})) & \& \\
\text{notin}(amount\_available, \text{supplier} : \text{monitorStock}(V+1, \text{part\_009})) & \& \; U < V.
\end{array}
$$

- Suppose there are n units of part_008 on stock and m units of part_009, where $n < m$.
- Then, $\mathsf{sol}(\chi^{(3)})$ is the singleton set containing the assignment $U{:=}n$, $V{:=}m$.
- However, if $n \geq m$, then $\mathsf{sol}(\chi^{(3)})$ is empty, since there are no satisfying assignments.

Integrity Constraints

An *integrity constraint* IC is an expression of the form

$$\psi \Rightarrow \chi$$

where:

- ψ is a safe code call condition, and
- χ is an atomic code call condition such that every root variable in χ occurs in ψ.

Example

$IC_2 :\ P = part\_008 \ \Rightarrow\ \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(3, \text{P}))$.

This IC says that at least three units of part_008 must always be available.

Example

$$
\begin{aligned}
IC_1 : \ & \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(\text{U}, \text{part\_001})) \ \& \\
& \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(\text{V}, \text{part\_002})) \\
& \qquad\qquad\qquad\qquad \Rightarrow \\
& \text{in}(amount\_available, \text{supplier} : \text{monitorStock}(\text{U} + \text{V}, \text{part\_008})).
\end{aligned}
$$

Is this an IC?

Example

This non-IC says says that the amount of *part_008* available is at least as high as the sume of the number of *part_001* available and the number of *part_002* 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 a 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 a least* 1.25 *times Rstk for all books published by XYZ Books.*

Integrity Constraint Satisfaction

- A state $\mathcal{O}_{\mathcal{S}}$ satisfies an integrity constraint IC of the form $\psi \Rightarrow \chi$, denoted $\mathcal{O}_{\mathcal{S}} \models IC$, if for every legal assignment of objects from $\mathcal{O}_{\mathcal{S}}$ to the variables in IC, either ψ is false or χ is true.
- $\mathcal{O}_{\mathcal{S}}$ satisfies a set of integrity constraints if it satisfies each integrity constraint in the set,

The is(,) Construct

- $\mathrm{is}(X, \mathcal{S} : \mathrm{f}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n}))$ is a code-call atom such that X is the set of all objects returned by the function f.
- $\mathrm{is}(X, \mathcal{S} : \mathrm{f}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n}))$ is in fact representable via an ordinary code call
- Associate with f, a function f_{set} such that:

$$f_{set}(d_1, \ldots, d_n) =_{def} \{\{o \mid \mathrm{in}(o, \mathcal{S} : \mathrm{f}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n}))\}\}.$$

- $\mathrm{is}(X, \mathcal{S} : \mathrm{f}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n}))$ is shorthand for the code call:

$$\mathrm{in}(X, \mathcal{S} : \mathrm{f_{set}}(\mathrm{d}_1, \ldots, \mathrm{d}_\mathrm{n})).$$