SPECIFYING AND VERIFYING CORRECTNESS OF TRIGGERS USING DECLARATIVE LOGIC PROGRAMMING – A FIRST STEP

Chitta Baral

Department of Computer Science & Engg
Arizona State University
Tempe, Arizona, USA
chitta@asu.edu
http://www.public.asu.edu/∼cbaral/

(joint work with Mutsumi Nakamura)

October 15, 2001

INTRODUCTION

Triggers and active databases

- Relational databases: a bunch of tables (relational instances)
- EMP

EmpId	EmpName	DeptName
27	John	Accounting
31	Mary	Administration
42	Peter	Services
51	Doug	Accounting
...	...	...

- DEPT

DeptName	ManagerId
Accounting	27
Service	34
...	...

- Triggers: Event-Condition-Action (ECA) Rules.

- Certain updates to the database trigger additional updates as dictated by the ECA rules.
- An ECA rule
  - Event: Deletion of a tuple in the EMP table
  - Condition: The EmpId in that tuple appears as a ManagerId in the DEPT table.
  - Action: Remove all such tuples in the DEPT table.

Current status: DB systems that have triggers and their usage

- Available in most recent database systems: IBM DB2/V2, Oracle, etc.
- But rarely used.
  - Too dangerous: automatically changes other tables.
  - What is the purpose of a set of triggers?
  - How do we state the purpose? In what langauge?
  - What does it mean that a set of triggers is not dangerous?
  - Correct! Correct with respect to whar?
  - Need to be able to specify the purpose.
  - Need to be able to formulate the notion of correctness of a set of triggers with respect to a specification.
  - Need to be able to verify the correctness.

– It would be great if certain triggers could be automatically generated from the specification.

SEMANTICS, SPECIFICATION, AND CORRECTNESS

Evolution of a database due to updates and triggers

- Updates and actions: Insert a tuple, delete a tuple, modify or update a tuple.
- Semantics: A function Ψ_T from states and action sequences to a sequence of states.
- $\Psi_T(\sigma, \alpha)$ is the sequence of database states recording how the database would evolve when a sequence of actions α is executed in σ in presence of the set of triggers T.
- Notations:
  - σ_α: denotes the last state of the evolution given by $\Psi(\sigma, \alpha)$.
  - $\sigma_{(\alpha_1,\alpha_2)}$: denotes the last state of the evolution given by $\Psi(\sigma_{\alpha_1}, \alpha_2)$
  - We similarly define $\sigma_{(\alpha_1,\ldots,\alpha_i)}$
  - $\sigma_\alpha, \sigma_{(\alpha_1,\alpha_2)}, \ldots$ is a sequence of **quiescent** states.

Specification ideas

- Four kinds: state invariance constraints; state maintenance constraints (or quiescent state constraints); trajectory invariance constraints; and trajectory maintenance constraints.
- Invariance vs maintenance: Invariance constraints are about all states of the database, while the maintenance constraints focus only on the quiescent states.
- A *state constraint* γ_s on a database scheme R, is a function that associates with each database r of R a boolean value $\gamma_s(r)$. A database r of R is said to *satisfy* γ_s if $\gamma_s(r)$ is true.
- A *trajectory constraint* γ_t on a database scheme R, is a function that associates with each database sequence Υ of R a boolean value $\gamma_t(\Upsilon)$. A database sequence Υ of R is said to *satisfy* γ_t if $\gamma_t(\Upsilon)$ is true.

- The specification in our example: "For any tuple t in the DEPT table, there must be a tuple t' in the EMP table such that $t.ManagerId = T'.EmpId$" is true in all quiescent states. (a trajectory maintenance constraint)

Definition of correctness

Let Γ_{si} be a set of state invariant constraints, Γ_{sm} be a set of state maintenance constraints, Γ_{ti} be a set of trajectory invariant constraints, Γ_{tm} be a set of trajectory maintenance constraints, A be a set of exogenous actions, and T be a set of ECA rules. We say T is correct with respect to $\Gamma_{si} \cup \Gamma_{sm} \cup \Gamma_{ti} \cup \Gamma_{tm}$ and A, if for all database states σ where the constraints in Γ_{si} and Γ_{sm} hold, and action sequences $\alpha_1, \ldots, \alpha_n$ consisting of exogenous actions from A,

- all the states in the sequences $\Psi(\sigma, \alpha_1)$, $\Psi(\sigma_{\alpha_1}, \alpha_2)$, $\ldots$, $\Psi(\sigma_{(\alpha_1,\ldots,\alpha_{n-1})}, \alpha_n)$ satisfy the constraints in Γ_{si};
- all the states $\sigma_{\alpha_1}, \ldots, \sigma_{(\alpha_1,\ldots,\alpha_n)}$ satisfy the constraints in Γ_{sm};
- the trajectory obtained by concatenating $\Psi(\sigma, \alpha_1)$ with $\Psi(\sigma_{\alpha_1}, \alpha_2)$, $\ldots$, $\Psi(\sigma_{(\alpha_1,\ldots,\alpha_{n-1})}, \alpha_n)$ satisfy the constraints in Γ_{ti}; and
- the trajectory $\sigma, \sigma_{\alpha_1}, \ldots, \sigma_{(\alpha_1,\ldots,\alpha_n)}$ satisfies the constraints in Γ_{tm}.

USING DLP FOR SPECIFICATION, SIMULATION AND VERIFICATION

Declarative Logic Programming (DLP)

- A DLP is a collection of rules of the form
  $a_0 \leftarrow a_1, \ldots, a_m, \textbf{not } a_{m+1}, \ldots, \textbf{not } a_n$
  where a_i's are atoms.
- Intuitive meaning: if $a_1 \ldots a_m$ are true and $a_{m+1} \ldots a_n$ can be assumed to be false then a_0 must be true.
- Semantics: Given in terms of answer sets.

An illustration: the database schema and the specification

- The schema

  $purchase(\underline{purchaseid}, client, amount)$.
  $payment(\underline{paymentid}, client, amount)$.
  $account(\underline{client}, credit, status)$.

  The underlined attributes are the primary keys and the attribute client in the relations *purchase* and *payment* is a foreign key with respect to the relation *account*. The relation *purchase* records the purchase history of clients and the relation *payment* records the payment history of clients. The relation *account* stores the available credit for each client and their credit status.

- Allowable exogenous actions: addition of tuples to the *purchase* and *payment* relations for existing clients.

- State maintenance constraints:

  1. For each client c which appears in a tuple a in the relation $account$: if $a.credit < 3K$ then $a.status = bad$, and if $a.credit \geq 3K$ then $a.status = good$.
  2. For each client c which appears in a tuple a in the relation $account$: $a.credit$ is $5K$ minus the sum of all the purchase amounts for c plus the sum of all the payment amounts for c.

- Triggers:
  - Trigger 1: When a tuple p is added to the $purchase$ relation, then the tuple a in the relation $account$ such that $p.client = a.client$ is updated so that the updated $a.credit$ has the value obtained by subtracting $p.amount$ from the old $a.credit$.
  - Trigger 2: When a tuple a in the relation $account$ is updated such that $a.credit$ is less than 3K then a is further updated such that $a.status$ has the value "bad".
  - Trigger 3: When a tuple p' is added to the $payment$ relation, then the tuple a in the relation $account$ such that $p'.client = a.client$ is updated so that the updated $a.credit$ has the value obtained by adding $p'.amount$ to the old $a.credit$.
  - Trigger 4: When a tuple a in the relation $account$ is updated such that $a.credit$ is more than or equal to 3K then a is further updated such that $a.status$ has the value "good".

A general methodology and an illustration

- Step 1: Representing the initial state (Π_{in})

  $holds(purchase(1, a, 3), 1).$
  $holds(purchase(2, b, 5), 1).$

  $holds(payment(1, a, 1), 1).$
  $holds(payment(2, b, 1), 1).$

  $holds(account(a, 3, good), 1).$
  $holds(account(b, 1, bad), 1).$
  $holds(account(c, 5, good), 1).$

- Step 1': Enumerating the possible initial states (Π_{in}^{enum}

$$
\begin{aligned}
&holds(purchase(X,Y,Z),1) \leftarrow \\
&\qquad id\_dom(X), cname\_dom(Y), amount(Z), \\
&\qquad \textbf{not } n\_holds(purchase(X,Y,Z),1). \\
&n\_holds(purchase(X,Y,Z),1) \leftarrow \\
&\qquad id\_dom(X), cname\_dom(Y), amount(Z), \\
&\qquad \textbf{not } holds(purchase(X,Y,Z),1). \\
&holds(payment(X,Y,Z),1) \leftarrow \\
&\qquad id\_dom(X), cname\_dom(Y), amount(Z), \\
&\qquad \textbf{not } n\_holds(payment(X,Y,Z),1). \\
&n\_holds(payment(X,Y,Z),1) \leftarrow \\
&\qquad id\_dom(X), cname\_dom(Y), amount(Z), \\
&\qquad \textbf{not } holds(payment(X,Y,Z),1). \\
&holds(account(X,Y,Z),1) \leftarrow \\
&\qquad cname\_dom(X), amount(Y), status(Z), \\
&\qquad \textbf{not } n\_holds(account(X,Y,Z).
\end{aligned}
$$

$$n\_holds(account(X,Y,Z),1) \leftarrow cname\_dom(X), amount(Y), status(Z), \textbf{not } holds(account(X,Y,Z).$$

$$\leftarrow maint\_constr(C), violated(C,1).$$

- Step 2: Action occurrence in the initial state (Π_{occ}):

$$occurs(ins, purchase(5,c,5),1).$$

- Step 2': Enumerating the initial exogenous actions (Π_{occ}^{enum})

$$not\_initially(X,Y) \leftarrow initially(U,V), U \neq X.$$
$$not\_initially(X,Y) \leftarrow initially(U,V), \textbf{not } same(Y,V).$$
$$initially(X,Y) \leftarrow possible(X,Y), \textbf{not } not\_initially(X,Y).$$

$$same(purchase(X,Y,Z), purchase(X,Y,Z)) \leftarrow id\_dom(X), cname\_dom(Y), amount(Z).$$
$$same(payment(X,Y,Z), payment(X,Y,Z)) \leftarrow id\_dom(X), cname\_dom(Y), amount(Z).$$
$$same(account(X,Y,Z), account(X,Y,Z)) \leftarrow cname\_dom(X), amount(Y), status(Z).$$

$occurs(X, Y, 1) \leftarrow initially(X, Y).$

- Step 3: Effect of actions and inertia (Π_{ef})

$holds(F, T+1) \leftarrow occurs(ins, F, T), executable(ins, F, T).$
$holds(G, T+1) \leftarrow occurs(upd, F, G, T), executable(upd, F, G, T).$

$ab(F, T+1) \leftarrow occurs(del, F, T), executable(del, F, T).$
$ab(F, T+1) \leftarrow occurs(upd, F, G, T), executable(upd, F, G, T).$

$holds(F, T+1) \leftarrow holds(F, T), occurred(T), \textbf{not}\ ab(F, T+1).$

- Step 4: Executability (Π_{ex})

$executable(ins, purchase(X, Y, W), T) \leftarrow.$
$executable(del, purchase(X, Y, W), T) \leftarrow holds(purchase(X, Y, W), T).$
$executable(ins, payment(X, Y, Z), T) \leftarrow.$
$executable(del, payment(X, Y, Z), T) \leftarrow holds(payment(X, Y, Z), T).$
$executable(ins, account(X, Y, Z), T) \leftarrow.$
$executable(del, account(X, Y, Z), T) \leftarrow holds(account(X, Y, Z), T).$

$executable(upd, account(X, Y, Z), account(X, Y, Z2), T) \leftarrow$
$holds(account(X, Y, Z), T).$
$executable(upd, account(X, Y, Z), account(X, Y2, Z), T) \leftarrow$
$holds(account(X, Y, Z), T).$

- Step 5: Trigers (Π_{tr})

$$occurs(del, account(Y, B, S), T + 1) \leftarrow holds(account(Y, B, S), T), occurs(ins, purchase(X, Y, W), T).$$

$$occurs(ins, account(Y, B - W, S), T + 1) \leftarrow holds(account(Y, B, S), T), occurs(ins, purchase(X, Y, W), T).$$

$$occurs(del, account(X, Y, S), T + 1) \leftarrow Y < 3, S = good, occurs(ins, account(X, Y, S), T).$$

$$occurs(ins, account(X, Y, bad), T + 1) \leftarrow Y < 3, S = good, occurs(ins, account(X, Y, S), T).$$

$$occurs(del, account(Y, B, S), T+1) \leftarrow holds(account(Y, B, S), T),$$
$$occurs(ins, payment(X, Y, W), T).$$

$$occurs(ins, account(Y, B+W, S), T+1) \leftarrow$$
$$holds(account(Y, B, S), T),$$
$$occurs(ins, payment(X, Y, W), T).$$

$$occurs(del, account(X, Y, S), T+1) \leftarrow Y \geq 3, S = bad,$$
$$occurs(ins, account(X, Y, S), T).$$

$$occurs(ins, account(X, Y, good), T+1) \leftarrow Y \geq 3, S = bad,$$
$$occurs(ins, account(X, Y, S), T).$$

- Step 6 : Identifying quiescent states (Π_{qu})

$$occurred(T) \leftarrow occurs(ins, F, T).$$
$$occurred(T) \leftarrow occurs(del, F, T).$$
$$occurred(T) \leftarrow occurs(upd, F, G, T).$$
$$occurs\_after(T) \leftarrow occurred(TT), T < TT.$$
$$quiescent(T+1) \leftarrow occurred(T), \textbf{not } occurs\_after(T).$$

- Step 7: Defining domains (Π_{dom})

$$fluent(purchase(X,Y,W)) \leftarrow$$
$$id\_dom(X), cname\_dom(Y), amount(W).$$
$$fluent(payment(X,Y,Z)) \leftarrow$$
$$id\_dom(X), cname\_dom(Y), amount(Z).$$
$$fluent(account(X,Y,Z)) \leftarrow$$
$$cname\_dom(X), amount(Y), status(Z).$$

- Step 8: Specification (Π_{cons})

$$payment\_total(C, Sum, T) \leftarrow time(T), cname\_dom(C), Sum[holds(payment(X,C,Y),T) : id\_dom(X) : amount(Y)]Sum.$$

$$purchase\_total(C, Sum, T) \leftarrow time(T), cname\_dom(C), Sum[holds(purchase(X,C,Y),T) : id\_dom(X) : amount(Y)]Sum.$$

$$\begin{aligned} violated(c2,T) \leftarrow\ & cname\_dom(C), payment\_total(C,Sum1,T),\\ & purchase\_total(C,Sum2,T), holds(account(C,Cr,Status),T),\\ & Cr \neq 5 - Sum2 + Sum1. \end{aligned}$$

$$\begin{aligned} violated(c1,T) \leftarrow\ & cname\_dom(C),\\ & holds(account(C,Cr,good),T), Cr < 3. \end{aligned}$$

$$\begin{aligned} violated(c1,T) \leftarrow\ & cname\_dom(C),\\ & holds(account(C,Cr,bad),T), Cr \geq 3. \end{aligned}$$

$$not\_correct \leftarrow maint\_constr(X), quiescent(T), violated(X,T).$$

$$correct \leftarrow \textbf{not}\ not\_correct.$$

$$weight \quad holds(payment(X,C,Y),T) = Y.$$
$$weight \quad holds(purchase(X,C,Y),T) = Y.$$

- Theorem 1:
  $\Pi_{in} \cup \Pi_{ef} \cup \Pi_{occ} \cup \Pi_{ex} \cup \Pi_{tr} \cup \Pi_{qu} \cup \Pi_{dom} \cup \Pi_{cons} \models_{smodels} correct$
- Theorem 2:
  $\Pi_{in}^{enum} \cup \Pi_{ef} \cup \Pi_{occ}^{enum} \cup \Pi_{ex} \cup \Pi_{tr} \cup \Pi_{qu} \cup \Pi_{dom} \cup \Pi_{cons} \models_{smodels} correct$

Next Steps: some in the paper

- More general constraints; temporal constructs.
- Various execution models. (deferred, immediate)
- Inferring events.
- More complicated triggers.
- Automatic generation of triggers. (before triggers; after triggers.)

Conclusion

- Active rules (triggers) are necessary in updating databases as operators do not necessarily know about the interrelationship associated with a database.
- But they may me dangerous unless we make sure that they are in some sense "correct".
- We discussed how to sepcify the purpose of a set of triggers; and what it means by a set of triggers to be correct with respect to a specification.
- We showed how to use declarative logic programming in simulating triggers and verifying their correctness.
- Exhaustive verification may take a long time, so we may do selective verification.