Automatic Software Testing

October 7th, 2009
CHR Working Week,
University of Ulm

Ralf Gerlich <ralf.gerlich@bsse.biz>

Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
Institut für Programmiermethodik und Compilerbau
89069 Ulm
http://www.informatik.uni-ulm.de/pm/

Dr. Rainer Gerlich System and Software Engineering
Auf dem Ruhbühl 181
88090 Immenstaad
http://www.bsse.biz/
Outline

- Automatic Test Data Generation
- Important Properties
- Applying CHR Theory (CHR, CHR^v, PCHR)
- Implementation Considerations
- Applying K.U. Leuven CHR Tools
- Conclusions
float sqrt(float a); /* to be tested */
void testsqrt() {
    for (int i=0; i<3000; i++) {
        float in = random();
        float out = sqrt(input);
        if (fabs(out*out - input) > epsilon)
            printf("failed, input %f\n", in);
    }
}

int gcd(int a, int b) {
    while (a!=b) {
        if (a>b) a = a-b;
        else b = b-a;
    }
    return a;
}

Large test case count, statistically significant results
Simple Stimulation ⇒ Small Computational Effort ⇒ Large Test Case Throughput
Oracle (if available)

"Abnormal" Input: P(a≤0 ∨ b≤0) ≈ 0.75
Corner Case: P(a=b) = 2^{-32} ≈ 2.3*10^{-10}

But:
Goal-directed deduction of test inputs required for some special cases
Pathconstraint: constraint, the solutions of which are the inputs leading to execution of a specific path in a program

Pathconstraint:
\[ T \cap \neg a \neq b \cap T \cap \neg a \geq b \cap a_1 = a - b \cap T \cap T \cap T \cap T \]

Solution:
\[ a \neq b \land a \geq b \land a_1 = a - b \land a_1 = b \iff a > b \land a = 2b \]
Infeasible paths

```c
void sort(int n, int[] a) {
    for (int i=0; i<n; i++) {
        int minElem = i;
        for (int j=i+1; j<n; j++)
            if (a[j] < a[minElem]) minElem = j;
        swap(a[i], a[minElem]);
    }
}
```

Runtime complexity: \( O(n^2) \)

Number of feasible paths with length \( \leq L \): \( o(\sqrt{L}) \)

Number of paths with length \( \leq L \): \( O(L^2 \cdot e^L) \)

The set of feasible paths grows considerably slower than set of paths in general!
Augmented Control-Flow Graphs

- Nodes and Edges describe possible control flow
- Execution of nodes modifies program state
- Selection of edges by a set of predicates
- Relational expression:
  - $x \text{ B(3) y}$
  - $x \text{ C(2,4) x}$
Path Constraints in Relational Calculus

\[ x \ (B\ (1) \ C\ (1,\ 2) \ B\ (2) \ C\ (2,\ 4) \ B\ (4) \ C\ (4,\ 5) \ B\ (5) \ C\ (5,\ 1) \ B\ (1) \ C\ (1,\ 6) \ B\ (6)) \ \ y \]
There is a path from node $a$ to node $b$, transforming input $x$ to output $y$. ("Specification")

There is a path from node $a$ to node $b$, with $u$ the output of node $a$ and $v$ the input of node $b$. ("Inner Specification")
Basic Theorems

Reduction to the Inner Specification

\[ \forall a, b, x, y : \quad a \neq b \Rightarrow \quad (x \ S_{a,b} \ y \iff x \ \B (a) \ I_{a,b} \ B (b) ) \ y) \]

\[ \forall a, b, x, y : \quad a = b \Rightarrow \quad (x \ S_{a,b} \ y \iff x \ \B (a) \ I_{a,b} \ B (b) ) \ y \cup x \ \B (a) \ y) \]

Recursive construction

\[ \forall a, b, x, y : \quad x \ I_{a,b} \ y \iff (a \rightarrow b \land x \ C (a, b) \ y) \lor \text{ Base Case} \]

\[ (\exists n : a \rightarrow^+ n \land n \rightarrow^+ b \land x \ (I_{a,n} \ B (n) I[n, b]) \ y) \text{ Recursive Case} \]

(split at some node n)

Path prediction

\[ \forall a, b, x, y, n : \quad \text{allpaths}(a, n, b) \Rightarrow (x \ I_{a,b} \ y \iff x \ (I_{a,n} \ B (n) I[n, b]) \ y) \]

All paths from a to b cross n
Important Properties

- **CHR declarative semantics**
  - Simplifies verification of implementation
  - Verification required: tool certification and confidence
  - Non-trivial for recursive construction

- **Non-Determinism**
  - Feasible paths not known in advance

- **Not Confluent**
  - Different executions $\Rightarrow$ different paths + inputs
  - We want it that way!

- **Usability for statistical evaluation**
  - Paths should be selected randomly
  - „Simple“ distribution
Transforming to \( \text{CHR}^v \)

**Reduction to the Inner Specification**

\[
x \ S_{[a,b]} \ y \iff a \neq b \ | x \ \mathcal{B}(a) \ u \land u \ \mathcal{I}_{[a,b]} \ v \land v \ \mathcal{B}(b) \ y \\
x \ S_{[a,b]} \ y \iff a = b \ | x \ \mathcal{B}(a) \ u \land u \ \mathcal{I}_{[a,b]} \ v \land v \ \mathcal{B}(b) \ y \lor x \ \mathcal{B}(a) \ y
\]

**Path prediction**

\[
x \ \mathcal{I}_{[a,b]} \ y \iff \text{allpaths}(a,n,b) \ | x \ \mathcal{I}_{[a,n]} \ u \land u \ \mathcal{B}(n) \ v \land v \ \mathcal{I}_{[n,b]} \ y
\]

**Recursive construction**

\[
x \ \mathcal{I}_{[a,b]} \ y \iff a \rightarrow b \ | x \ \mathcal{C}(a,b) \ y \\
x \ \mathcal{I}_{[a,b]} \ y \iff a \rightarrow^* n \land n \rightarrow^+ b \ | x \ \mathcal{I}_{[a,n]} \ u \land u \ \mathcal{B}(n) \ v \land v \ \mathcal{I}_{[n,b]} \ y
\]

- **Declarative Semantics incorrect**
  - LHS and RHS not equivalent
  - No failure if there is no path from \( a \) to \( b \)
- **Split-node is free variable**
Correct Approach with CHR^v and Guards

Recursive reduction

\[ x \mathcal{I}_{[a,b]} y \iff a \not\rightarrow^* b \perp \Rightarrow \text{Trivial Fail} \]

\[ x \mathcal{I}_{[a,b]} y \iff a \rightarrow^* b \mid x \mathcal{I}^1_{[a,b]} y \lor x \mathcal{I}^{>1}_{[a,b]} y \]

\[ x \mathcal{I}^1_{[a,b]} y \iff a \rightarrow b \mid x \mathcal{C}(a, b) y \Rightarrow \text{Length} = 1 \]

\[ x \mathcal{I}^{>1}_{[a,b]} y \iff a \rightarrow^+ n \land n \rightarrow^+ b \mid x \mathcal{I}_{[a,n]} u \land u \mathcal{B}(n) v \land v \mathcal{I}_{[n,b]} y \Rightarrow \text{Length} > 1 \]

- Introduced additional constraints
  - Logical reading based on length of path
- New rule for trivial fail
  - Logical reading: There is no path from a to b at all
- Split-node is all-quantified, but not part of search
  - Graph-Relations as user-defined constraints?
- Verification more difficult than for other rules
Probabilistic CHR

- Rules are annotated with weight
- On each step, collect applicable rule instances
- Choose one randomly, based on weights
- Source-to-Source-Transformation in CHR
  - Uses non-standard remove_constraint/1
  - No support by theory
Applying PCHR

- There is no PCHR\textsuperscript{v}
  - Problem for non-confluent solvers

- Classical search insufficient
  - Paths need to be selected randomly
  - BFS, DFS lead to deterministic path selection or infinite recursion

- Ad-Hoc PCHR\textsuperscript{v}
  - Consider all branches for applicable rule instances

- But: No explicit weights between branches
  - For example: base case vs. recursion case
  - Base case: one instance
  - Recursion case: one instance / split node
  - Priority for recursion case increases with number of branches
Alternative Application

Apply pure PCHR (no search)

\[ x \mathcal{I}_{[a,b]} y \iff a \rightarrow b | x \mathcal{C} (a,b) \ y \]

\[ x \mathcal{I}_{[a,b]} y \iff a \rightarrow^+ n \land n \rightarrow^+ b | x \mathcal{I}_{[a,n]} u \land u \mathcal{B}(n) \land v \land v \mathcal{I}_{[n,b]} y \]

- Individual weights for base- and recursion-case
  - Weight for recursion still depends on split-nodes

- Semantics
  - Operational: correct ✓
  - Declarative: incorrect ✗

- CHR advantages lost
  - Search
  - Verification
Another try

Recursion-weight independent of split-nodes

Semantics
- Operational: correct ✓
- Declarative: incorrect ✗

Additional guard
- Operational overhead by duplicate check
- Required to avoid selection of inapplicable rules

Probability distribution
- Constraints without direct solution have lower weight

There is a path from a to b with length>1

Reuse additional constraints

\[ x \ I_{[a,b]} y \iff a \not\rightarrow^* b \perp \]
\[ x \ I_{[a,b]} y \iff a \rightarrow b \mid x \ I_{[a,b]}^1 y \]
\[ x \ I_{[a,b]} y \iff a \rightarrow^{>1} b \mid x \ I_{[a,b]}^{>1} y \]
Actual implementation

• Handcrafted Random Selection
  – Select open path (random, uniform distribution)
  – Select base- or recursion case (weights $1-p : p$)
  – Alternative case is used for search
  – For recursion case, select split node
  – Exhaustively apply prediction

• Predictable Distribution
  – Mean path length depends on $p$
  – Still only an approximation (infeasible paths)
  – Termination probability may be $<1$
  – Replaced recursion by linear alternative (step)
  – Added node-weights, controlling loop entry / exit

• Issues
  – Verification very difficult
  – „Bad hack“: nonsense declarative semantics
Example Distribution
Implementation Considerations

- **Buitin-Solver**
  - FD-Solver is not good at detecting contradictions
  - e.g. $a < b \land b < a$ leads to long step-by-step domain reduction
  - Solution: Combine Axioms with FD-rules
  - Implemented in SWI CHR (11 constraints, 80 rules)
  - e.g. $a < b \land b < a \Rightarrow \bot$.
  - e.g. $a + b = c \land a + b = d \iff c = d$.

- **Emulation of Memory-Access**
  - Pointers and Arrays of arbitrary depth (e.g. C, C++)
  - $\text{read(State,Ref,Var)}$
  - $\text{write(NewState,OldState,Ref,Var)}$
  - Requires efficient lookup in constraint store
The K.U. Leuven Compiler System

- Optimising compiler
  - Elimination of passive / dead rules
  - Compression of successive rule applications
  - Optimised constraint store lookup

- Static Analysis and Checks
  - „By-product“ of optimisation
  - Warnings about dead rules
  - Detection of infeasible guards
  - Type system
  - Important hints to bugs!
  - To be extended (infinite recursion, ...)

Copyright © 2009, Ralf Gerlich. All rights reserved.
K.U. Leuven Java CHR

• Optimised CHR-to-Java-Compiler
  – K.U. Leuven CHR features (static analysis, ...)
  – Our FD-Solver largest JCHR program to date
• Easy integration with Java-core of test tool
• Bugs, but...
  – ...fast feedback and fixes from Peter van Weert
• But: No Search
  – Principal problem with Java
• Tried to add trailing
  – „Moving target“
  – Lack of design documentation
  – Gave up after ½ year and moved to SWI
Conclusions

● **CHR assets**
  - Matching declarative and operational semantics
  - Diversity of Tools

● **Difficulties**
  - search theory+practice: free variables, alternative rules
  - PCHR: fine-grained control of probabilities

● **Priorities (from user POV)**
  - Integration of different theories (PCHR, CHR\textsuperscript{v}, refined semantics)
  - Optimisation potential
  - Static Checking
  - Search in non-Prolog-based environments

● **Non-Priorities**
  - Status as general-purpose language
Questions?
Backup
Number of paths of length $l$ starting at node $k$

$$n_k(l) = \begin{cases} 1 & \text{for } k = e \\ 0 & \text{else} \end{cases}$$

$$n_k(l) = \sum_{s \in \text{pred}(k)} n_s(l-1)$$

Randomly select a path (uniform distribution over paths of length $L \leq N$)

then construct path constraint

No solution? Next try!

Large share of infeasible paths makes algorithm impracticable!
int ggt(int a, int b) {
    while (a!=b) {
        if (a>b) a=a-b;
        else b=b-a;
    }
    return a;
}

ggt(A1,B1,Ret):-
    while(A2!=B2, (A1,B1), (A2,B2), (A3,B3),
    Ret=A3.

Cin ⇒ (while(C,Vin,V,Vout,B) → Bin,
        while(C,Vin',V,Vout,B'))
mit
Bout = subst(B,Vout→Vin')
Cin = subst(C,V→Vin)
B' = subst(B,Vin→Vin')
Sequential construction of path constraint with backtracking

Specific successors are favoured:

- Control-Dependencies of goal nodes
- Loop-ending nodes

No parallel handling of sub-paths

⇒

No prediction of infeasible paths
CHR [Constraint Handling Rules] are essentially a committed-choice language consisting of multi-headed guarded rules that rewrite constraints into simpler ones until they are solved."

Thom Frühwirth, 1994

Simplification: \( H \iff G \mid C \).

Propagation: \( H \Rightarrow G \mid C \).

Simpagation: \( H1 \setminus H2 \iff G \mid C \).

H User-defined Constraints

G Builtin Constraints

C User-defined+Builtin Constraints

Operationelle Semantik: Regeln \( \rightarrow \) Termersetzungen

Deklarative Semantik: Regeln \( \rightarrow \) Äquivalenzaussagen