Automatic Detection and Repair of Errors in Data Structures

Brian Demski, Martin Rinard

April 28, 2005

Motivation

The problem

- Programs manipulate data structures
- Software error or anomaly causes inconsistency
- Assumptions under which software was developed no longer hold
  - software behaves in unpredictable manner

The solution proposed

- Do *not* increase the reliability of the code
- Specify key data structure consistency constraints
- Dynamically detect and repair data structures violating the constraints

Motivation

The problem

- Programs manipulate data structures
- Software error or anomaly causes inconsistency
- Assumptions under which software was developed no longer hold
  - software behaves in unpredictable manner

The solution proposed

- Do *not* increase the reliability of the code
- Specify key data structure consistency constraints
- Dynamically detect and repair data structures violating the constraints

Goal

- Do not necessarily restore the data structure to previous consistent state the program was into
- Deliver repaired data structures satisfying the consistency assumptions of the program
  - The program is able to continue to operate successfully

Intended Scope

- Prioritize continued execution even after concrete evidence of error
- Clearly might not be acceptable for all computations
  - *safety-critical systems* like air traffic control can benefit

Goal

- Do not necessarily restore the data structure to previous consistent state the program was into
- Deliver repaired data structures satisfying the consistency assumptions of the program
  - The program is able to continue to operate successfully

Intended Scope

- Prioritize continued execution even after concrete evidence of error
- Clearly might not be acceptable for all computations
  - *safety-critical systems* like air traffic control can benefit

Basic Approach

Data structure views

- Concrete view – in memory bits
- Abstract view – relations between abstract objects
  - specification of high level constraints
  - reasoning to repair inconsistencies

Specification

- Set of model definition rules
- Set of consistency constraints

Automatic tool

- Generate algorithm that builds the model,
- Inspect the model and data structures to find constraint violations
- Repair data structures

Repair algorithm

- Inconsistency detection
- Converts each violated constraint to DNF (disjunctive normal form)
- Apply repair actions – may cause other constraint to be violated

Automatic tool

- Generate algorithm that builds the model,
- Inspect the model and data structures to find constraint violations
- Repair data structures

Repair algorithm

- Inconsistency detection
- Converts each violated constraint to DNF (disjunctive normal form)
- Apply repair actions – may cause other constraint to be violated

Invoking Check & Repair

- Data structures are updated, may be inconsistent temporarily
- Programmer marks program points where he/she expects data structures to be consistent
- Augment programs to perform check & repair in signal handlers
- Persistent data structures – use a stand alone separate mechanism

An example, FAT

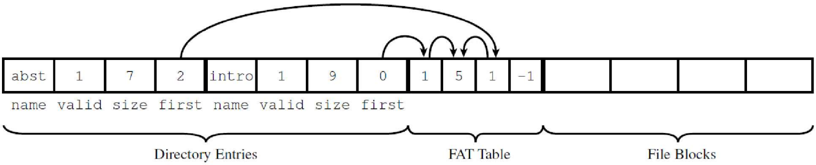

Figure: Inconsistent File System

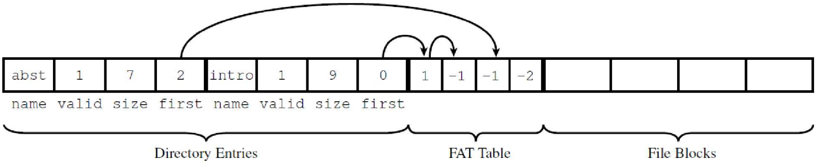

Figure: Repaired File System

FAT Constraints

- **Chain Disjointness:** Each block should be in at most one chain
- **Free Block Consistency:** No chain should contain a block marked as free

Abstract Constraints

- Developer specifies a translation from concrete data structure representation to abstract model
- Express the constraints in terms of the abstract model

FAT Constraints

- **Chain Disjointness:** Each block should be in at most one chain
- **Free Block Consistency:** No chain should contain a block marked as free

Abstract Constraints

- Developer specifies a translation from concrete data structure representation to abstract model
- Express the constraints in terms of the abstract model

Object and Relation Declarations

```
set blocks of integer : partition used | free;
        relation next : used -> used;
```

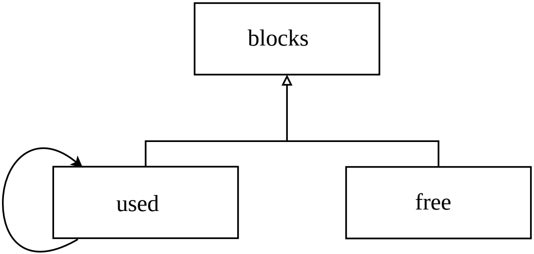

Figure: Graphical Representation of Object and Relation Declarations

```
struct Entry {
  byte name[Length];
  byte valid;
  int  size;
  int  first;
}
struct Block { data byte[BlockSize]; }
struct Disk {
  Entry table[NumEntries];
  int FAT[NumBlocks];
  Block block[NumBlocks];
}
```

Figure: Structure Declarations

```
Disk disk;

for i in 0..NumEntries, disk.table[i].valid &&
  disk.table[i].first < NumBlocks =>
  disk.table[i].first in used;
for b in used, 0 <= disk.FAT[b] &&
  disk.FAT[b] < NumBlocks => disk.FAT[b] in used;
for b in used, 0 <= disk.FAT[b] &&
  disk.FAT[b] < NumBlocks =>
  <b,disk.FAT[b]> in next;
for b in 0..NumBlocks, !(b in used) => b in free;
```

Figure: Model Definition Declarations and Rules

Rule structure

- quantifier identifying the scope of the rule
- guard that has to be true for the rule to apply
- inclusion constraint – used to build the sets & relations

Internal Consistency Constraints

- *Internal* constraints are stated using the model exclusively
  - Do not use the concrete data structures

```
for b in used, size(next.b) <= 1;
```

Inconsistent Model   Repaired Model

External Consistency Constraints

- May reference both model and concrete data structures
- Captures requirements the sets and relations place on the values in the concrete data structures
- Used to translate model repairs back into concrete data structure

```
for b in free, disk.FAT[b] = -2;
for <i, j> in next, disk.FAT[i] = j;
for b in used, size(b.next) = 0 => disk.FAT[b] = -1;
```

Specification Language

Sublanguages

- Structure definition language
- Model definition language
- Language for constraints
  - internal constraints
  - external constraints

Structure Definition Language

Declaring the layout of data in memory

$$
\begin{aligned}
structdefn &:= \texttt{struct}\ structurename \\
&\quad (subtypes\ structurename)\ \{fielddefn^*\} \\
fielddefn &:= type\ field; \mid \texttt{reserved}\ type; \mid \\
&\quad type\ field[E]; \mid \\
&\quad \texttt{reserved}\ type[E]; \\
type &:= \texttt{boolean} \mid \texttt{byte} \mid \texttt{short} \mid \texttt{int} \mid structurename \\
&\quad structurename\ * \\
E &:= V \mid number \mid string \mid E.field \mid \\
&\quad E.field[E] \mid E - E \mid E + E \mid E/E \mid E * E
\end{aligned}
$$

Figure: Structure Definition Language

Model Definition Language

- Define a translation from concrete data structures into an abstract model
- Set declaration `set S of T : partition` S_1, ..., S_n
  - `partition` keyword can be replaced by `subsets`
- Relation declaration `relation R:` S_1 `->` S_2

Given a model containing the rules, set of concrete data structures *h*, naming environment *l*, the model is the least fixed point of the functional:

$$\lambda m.(\mathcal{R}[C_1]\ h\ l) \ldots (\mathcal{R}[C_1]\ h\ l\ m)$$

Model Definition Language

$$\begin{array}{lcl} C & := & Q, C \mid G \Rightarrow I \\ Q & := & \texttt{for } V \texttt{ in } S \mid \texttt{for } \langle V, V\rangle \texttt{ in } R \mid \\ & & \texttt{for } V = E \texttt{ .. } E \\ G & := & G \texttt{ and } G \mid G \texttt{ or } G \mid !G \mid E = E \mid E < E \mid \texttt{true} \mid \\ & & (G) \mid E \texttt{ in } S \mid \langle E, E\rangle \texttt{ in } R \\ I & := & E \texttt{ in } S \mid \langle E, E\rangle \texttt{ in } R \\ E & := & V \mid number \mid string \mid E.field \mid \\ & & E.field[E] \mid E - E \mid E + E \mid E/E \mid E * E \end{array}$$

Figure: Model Definition Language

Internal Constraints

- Each constraint is a sequence of quantifiers $Q_1, Q_2, \ldots, Q_n$ followed by body B
- B uses logical connectors (i.e. `and`, `or`, `not`) to combine basic propositions P
- Constraint C is evaluated in the context of a model m and a naming environment l

Complication

For undefined values cannot yield `true` or `false` $\Rightarrow$ extend to 3-value logic by introducing `maybe`

Internal Constraints

- Each constraint is a sequence of quantifiers $Q_1, Q_2, ..., Q_n$ followed by body B
- B uses logical connectors (i.e. `and`, `or`, `not`) to combine basic propositions P
- Constraint C is evaluated in the context of a model m and a naming environment l

Complication

For undefined values cannot yield `true` or `false` $\Rightarrow$ extend to 3-value logic by introducing `maybe`

Internal Constraints

$$
\begin{array}{rcl}
C & := & Q, C \mid B \\
Q & := & \texttt{for } V \texttt{ in } S \mid \texttt{for } V = E \texttt{ .. } E \\
B & := & B \texttt{ and } B \mid B \texttt{ or } B \mid !B \mid (B) \mid \\
 & & VE = E \mid VE < E \mid VE <= E \mid VE > E \mid \\
 & & VE >= E \mid V \texttt{ in } SE \mid \texttt{size}(SE) = C \mid \\
 & & \texttt{size}(SE) >= C \mid \texttt{size}(SE) <= C \\
VE & := & V.R \\
E & := & V \mid number \mid string \mid E + E \mid E - E \mid E/E \mid \\
 & & E * E \mid E.R \mid \texttt{size}(SE) \mid (E) \\
SE & := & S \mid V.R \mid R.V
\end{array}
$$

Figure: Internal Constraints Language

External Constraints

Each constraint has

- a quantifier identifying the scope of the rule
- a guard G that must be true for the constraint to apply
- a condition C speciffying the value of either
  - a program variable
  - a field in a structure
  - an array element
- Constraint R evaluated in the context of naming environment l, model m and set of concrete data structures h

External Constraints

$$
\begin{array}{rcl}
R & := & Q, R \mid G \Rightarrow C \\
Q & := & \texttt{for}\ V\ \texttt{in}\ S \mid \texttt{for}\ \langle V, V\rangle\ \texttt{in}\ R \mid \texttt{for}\ V = E\ ..\ E \\
G & := & G\ \texttt{and}\ G \mid G\ \texttt{or}\ G \mid !G \mid E = E \mid E < E \mid \texttt{true} \\
C & := & HE.field = E \mid HE.field[E] = E \mid V = E \\
HE & := & V \mid HE.field \mid HE.field[E] \\
E & := & V \mid number \mid string \mid E.R \mid E - E \mid E + E \mid \\
 & & E * E \mid E/E \mid \texttt{size}(SE) \mid \texttt{element}\ E\ \texttt{of}\ SE \\
SE & := & S \mid V.R \mid R.V
\end{array}
$$

Figure: External Constraints Language

Error Detection and Repair

Detection

Detect violations by evaluating constraints (internal & external) in the context of the model

- Iterates over all values of the quantified variables, evaluating body

Repair

Updates the model and the concrete data structures $\Rightarrow$ all internal & external constraints satisfied

- Repair actions coerce propositions to be true

Error Detection and Repair

Detection

Detect violations by evaluating constraints (internal & external) in the context of the model

- Iterates over all values of the quantified variables, evaluating body

Repair

Updates the model and the concrete data structures ⇒ all internal & external constraints satisfied

- Repair actions coerce propositions to be true

Error Detection & Repair in Internal Phase

Repair Algorithm

- Input: a body and variable bindings that falsify the body
- Output: change the model to make the body true
  - Converts body to DNF
  - Each basic proposition has an associated repair action
  - Choose one conjunction, apply repair to all basic propositions

Repair Actions

Size propositions

- `size`$(S) = C$, `!size`$(S) = C$, `size`$(S) > C \ldots$
- C integer constant, S set or relation expr ($R.v$ or $v.R$)
- S is set $\Rightarrow$ add or remove items to the set
  - make sure the `partition` and subset inclusion are preserved
- S is a relation expr $\Rightarrow$ add or remove tuples
  - may modify domains

Where do items come from?

- `structs` – memory allocation primitives; supersets
- primitive types – synthesize new values

Repair Actions

Size propositions

- $\texttt{size}(S) = C$, $!\texttt{size}(S) = C$, $\texttt{size}(S) > C \dots$
- C integer constant, S set or relation expr ($R.v$ or $v.R$)
- S is set $\Rightarrow$ add or remove items to the set
    - make sure the `partition` and subset inclusion are preserved
- S is a relation expr $\Rightarrow$ add or remove tuples
    - may modify domains

Where do items come from?

- `structs` – memory allocation primitives; supersets
- primitive types – synthesize new values

Repair Actions

Inequality Propositions

- $V.R = E$, $!V.R = E$, $V.R < E$, $V.R <= E$, $V.R > E$, $V.R >= E$
- Evaluate E, update $V.R$

Inclusion Propositions

- V in SE, where SE is a set in the model or a relation expression
- Add or remove value referenced by V, obeying partition and subset requirements

Repair Actions

Inequality Propositions

- $V.R = E$, $!V.R = E$, $V.R < E$, $V.R <= E$, $V.R > E$, $V.R >= E$
- Evaluate E, update $V.R$

Inclusion Propositions

- V `in` SE, where SE is a set in the model or a relation expression
- Add or remove value referenced by V, obeying partition and subset requirements

Choosing the Conjunction to Repair

- When faced with a choice – use a cost heuristic
  - additive cost function for the repair actions of each proposition in a conjunction
- Designed to minimize the number of changes
- Tuned to discourage removal of objects
  - preserve as much information about original data structures as possible

Termination

Constraint dependence graph

- One node for every constraint and one node for every conjunction in DNF
- Edges:
  - Constraint to Conjunctions
  - Interference
  - Quantifier Scope

acyclic graph ? will terminate : prune conjunctions & and try again

Error Detection & Repair in External Phase

Repair

- Detection algorithm ⇒ variable bindings that falsify constraint
- Assign data structure value to be same as model value

Methodology

Implementation

Interpreter to construct model, check for consistency violations and repair

Test subjects

- CTAS – air-traffic control tools
- Simplified version of `ext2`
- Freeciv – multiplayer game
- Microsoft Office File Format

Performance

Application	Number of model definition rule applications	Total size of sets (objects)	Total size of relations (tuples)
CTAS	20	8	2
File system	11720	3128	1954
Freeciv	63072	7537	15990
Word	139740	64	17

Table: Number of model rule applications and size of model

Performance

Application	Time to construct model (ms)
CTAS	4.2
File system	1,188.9
Freeciv	5,609.1
Word	7,189.5

Table: Time to construct model

Performance

Application	Internal constraint evaluations	Time to check internal constraints (ms)
CTAS	4	0.09
File system	2384	16.6
Freeciv	16004	175.3
Word	28	0.2

Table: Number of checks and time to check and repair internal constraints

Performance

Application	External constraint evaluations	Time to enforce external constraints (ms)
CTAS	4	0.2
File system	3164	59.5
Freeciv	12001	171.4
Word	39	1.2

Table: Number of checks and time to enforce external constraints

Fin