# Automatic Software Testing

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

Ralf Gerlich <ralf.gerlich@bsse.biz>

| | |
|---|---|
| Universität Ulm<br>Fakultät für Ingenieurwissenschaften und Informatik<br>Institut für Programmiermethodik und Compilerbau<br><br>89069 Ulm<br><br>http://www.informatik.uni-ulm.de/pm/ | Dr. Rainer Gerlich System and Software Engineering<br><br>Auf dem Ruhbühl 181<br><br>88090 Immenstaad<br><br>http://www.bsse.biz/ |

# Outline

- Automatic Test Data Generation
- Important Properties
- Applying CHR Theory (CHR, $CHR^{\vee}$, PCHR)
- Implementation Considerations
- Applying K.U. Leuven CHR Tools
- Conclusions

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Random Test

```
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);
    }
}
```

Large test case count, statistically significant results

Simple Stimulation ⇒ Small Computational Effort ⇒ Large Test Case Throughput

Oracle (if available)

(taken from: "Random Testing", R. Hamlet, 1994)

**But:**

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

"Abnormal" Input:
$P(a \leq 0 \vee b \leq 0) \approx 0{,}75$

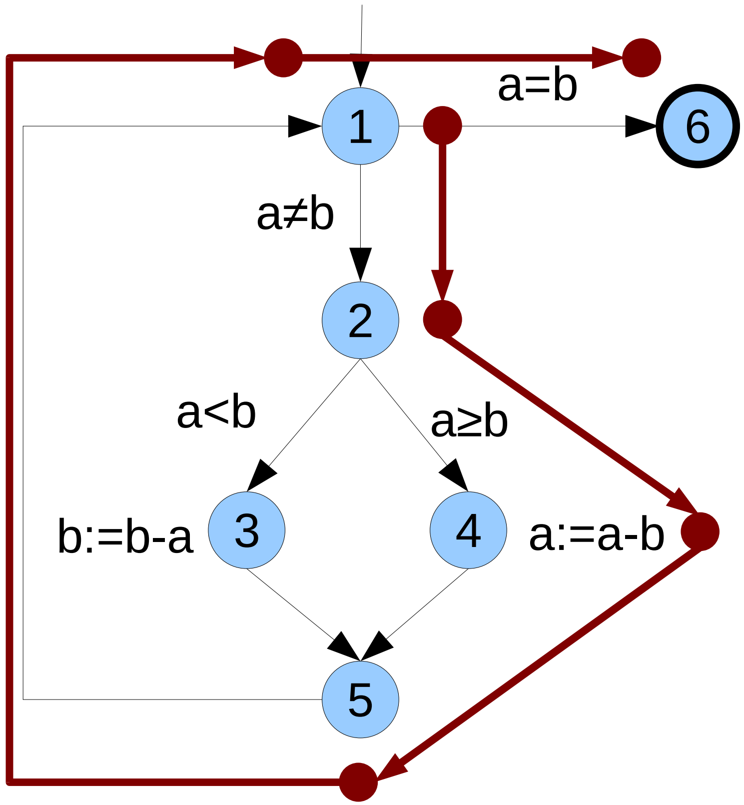
Corner Case:
$P(a=b) = 2^{-32} \approx 2{,}3 \cdot 10^{-10}$

**Goal-directed deduction of test inputs required for some special cases**

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Pathconstraints

**Pathconstraint: constraint, the solutions of which are the inputs leading to execution of a specific path in a program**




**Pathconstraint:**

$\top$

$\wedge a \neq b$

$\wedge \top$

$\wedge a \geq b$

$\wedge a_1 = a - b$

$\wedge \top \wedge \top \wedge \top \wedge \top$

$\wedge a_1 = b$

$\wedge \top$

**Solution:**

$$a \neq b \wedge a \geq b \wedge a_1 = a - b \wedge a_1 = b$$

$$\Leftrightarrow$$

$$a > b \wedge a = 2b$$

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Infeasible paths

```
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]);
    }
}
```

Inner loop depends on outer loop

Runtime complexity: $O(n^2)$

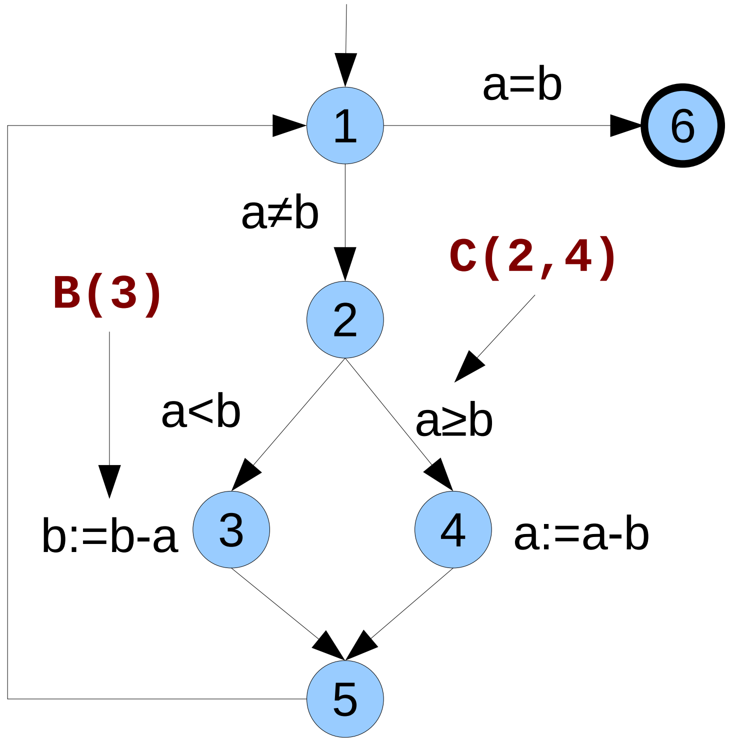
Number of feasible paths with length≤L: $o(\sqrt{L})$

Number of paths with length≤L: $O(L^2 * e^L)$

**The set of feasible paths grows considerably slower than set of paths in general!**

Copyright © 2009, Ralf Gerlich. All rights reserved.

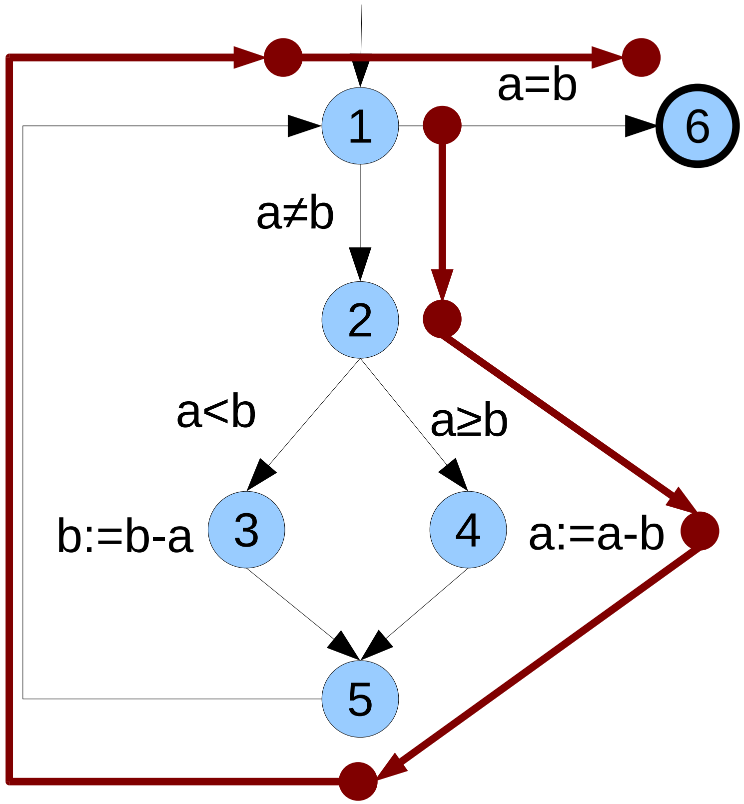
# Augmented Control-Flow Graphs




- Nodes and Edges describe possible control flow
- Execution of nodes modifies program state
- Selection of edges by a set of predicates
- Relational expression:
  - `x B(3) y`
  - `x C(2,4) x`

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Path Constraints in Relational Calculus




$$x \;\; (\mathcal{B}\,(1)\,\mathcal{C}\,(1,2)\,\mathcal{B}\,(2)\,\mathcal{C}\,(2,4)\,\mathcal{B}\,(4)\,\mathcal{C}\,(4,5)\,\mathcal{B}\,(5)\,\mathcal{C}\,(5,1)\,\mathcal{B}\,(1)\,\mathcal{C}\,(1,6)\,\mathcal{B}\,(6)) \;\; y$$

Copyright © 2009, Ralf Gerlich. All rights reserved. 

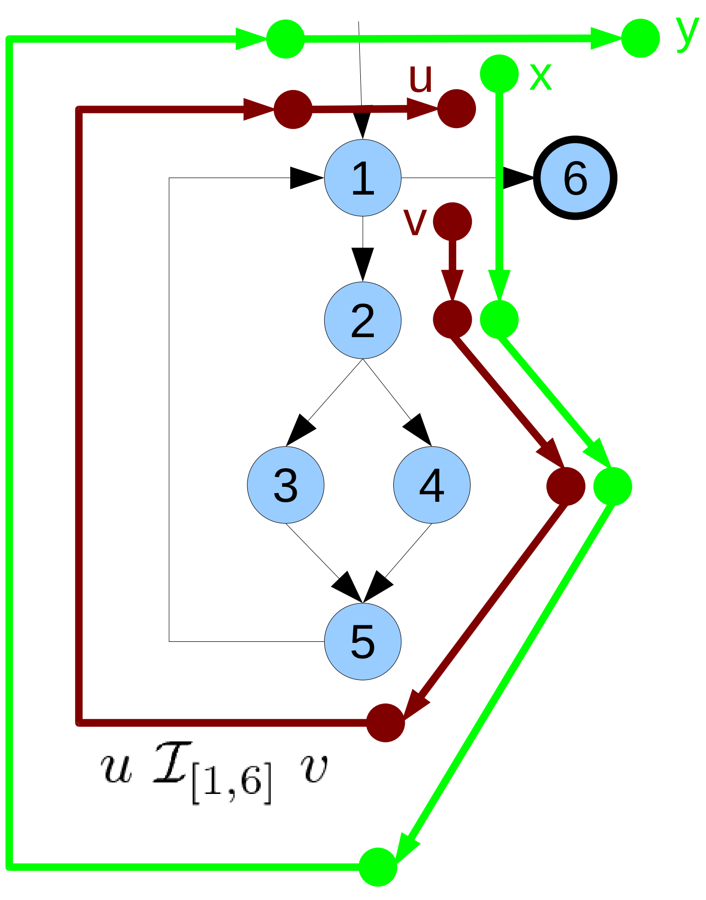
# Generic Path Constraint Relations

$$x \; \mathcal{S}_{[a,b]} \; y$$

There is a path from node a to node b, transforming input x to output y. ("Specification")

$$u \; \mathcal{I}_{[a,b]} \; v$$

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")




$u \; \mathcal{I}_{[1,6]} \; v$

$x \; \mathcal{S}_{[1,6]} \; y$

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Basic Theorems

## Reduction to the Inner Specification

$$\forall a,b,x,y: \quad a \neq b \Rightarrow \quad \left(x\ \mathcal{S}_{[a,b]}\ y \Leftrightarrow x\ \left(\mathcal{B}(a)\,\mathcal{I}_{[a,b]}\mathcal{B}(b)\right)\ y\right)$$

$$\forall a,b,x,y: \quad a = b \Rightarrow \quad \left(x\ \mathcal{S}_{[a,b]}\ y \Leftrightarrow x\ \left(\mathcal{B}(a)\,\mathcal{I}_{[a,b]}\mathcal{B}(b)\right)\ y \cup x\ \mathcal{B}(a)\ y\right)$$

## Recursive construction

$$\forall a,b,x,y: x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad (a \rightarrow b \wedge x\ \mathcal{C}(a,b)\ y) \vee$$

Base Case

$$\left(\exists n: a \rightarrow^{+} n \wedge n \rightarrow^{+} b \wedge x\ \left(\mathcal{I}_{[a,n]}\mathcal{B}(n)\,\mathcal{I}[n,b]\right)\ y\right)$$

Recursive Case
(split at some node n)

## Path prediction

$$\forall a,b,x,y,n: allpaths(a,n,b) \Rightarrow \left(x\ \mathcal{I}_{[a,b]}\ y \Leftrightarrow x\ \left(\mathcal{I}_{[a,n]}\mathcal{B}(n)\,\mathcal{I}_{[n,b]}\right)\ y\right)$$

All paths from a to b cross n

Copyright © 2009, Ralf Gerlich. All rights reserved.

# 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 ⇒ different paths + inputs
  - We want it that way!
- Usability for statistical evaluation
  - Paths should be selected randomly
  - „Simple“ distribution

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Transforming to CHR$^\vee$

## Reduction to the Inner Specification

$$x\ \mathcal{S}_{[a,b]}\ y \quad \Leftrightarrow \quad a \neq b | x\ \mathcal{B}(a)\ \ u \wedge u\ \mathcal{I}_{[a,b]}\ v \wedge v\ \mathcal{B}(b)\ \ y$$

$$x\ \mathcal{S}_{[a,b]}\ y \quad \Leftrightarrow \quad a = b | x\ \mathcal{B}(a)\ \ u \wedge u\ \mathcal{I}_{[a,b]}\ v \wedge v\ \mathcal{B}(b)\ \ y \vee x\ \mathcal{B}(a)\ \ y$$

## Path prediction

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad allpaths(a,n,b) | x\ \mathcal{I}_{[a,n]}\ u \wedge u\ \mathcal{B}(n)\ \ v \wedge v\ \mathcal{I}_{[n,b]}\ y$$

## Recursive construction

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

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad a \rightarrow^{+} n \wedge n \rightarrow^{+} b | x\ \mathcal{I}_{[a,n]}\ u \wedge u\ \mathcal{B}(n)\ \ v \wedge v\ \mathcal{I}_{[n,b]}\ y$$

- Declarative Semantics incorrect
  - LHS and RHS not equivalent
  - No failure if there is no path from a to b
- Split-node is free variable

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Correct Approach with CHR$^\vee$ and Guards

## Recursive reduction

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad a \not\rightarrow^* b | \bot$$

Trivial Fail

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad a \rightarrow^* b | x\ \mathcal{I}^{1}_{[a,b]}\ y \vee x\ \mathcal{I}^{>1}_{[a,b]}\ y$$

$$x\ \mathcal{I}^{1}_{[a,b]}\ y \quad \Leftrightarrow \quad a \rightarrow b | x\ \mathcal{C}\left(a,b\right)\ y$$

Length = 1

$$x\ \mathcal{I}^{>1}_{[a,b]}\ y \quad \Leftrightarrow \quad a \rightarrow^+ n \wedge n \rightarrow^+ b | x\ \mathcal{I}_{[a,n]}\ u \wedge u\ \mathcal{B}\left(n\right)\ v \wedge v\ \mathcal{I}_{[n,b]}\ y$$

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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Probabilistic CHR

- Frühwirth, Di Pierro, Wiklicky: Probabilistic Constraint Handling Rules, 2002
- 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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Applying PCHR

- There is no PCHR$^{v}$
  - 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$^{v}$
  - 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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Alternative Application

## Apply pure PCHR (no search)

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad a \longrightarrow b | x\ \mathcal{C}\,(a,b)\ \ y$$

$$x\ \mathcal{I}_{[a,b]}\ y \quad \Leftrightarrow \quad a \longrightarrow^{+} n \wedge n \longrightarrow^{+} b | x\ \mathcal{I}_{[a,n]}\ u \wedge u\ \mathcal{B}\,(n)\ \ v \wedge v\ \mathcal{I}_{[n,b]}\ y$$

- 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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Another try

$$x \; \mathcal{I}_{[a,b]} \; y \quad \Leftrightarrow \quad a \not\to^* b | \bot$$

$$x \; \mathcal{I}_{[a,b]} \; y \quad \Leftrightarrow \quad a \to b | x \; \mathcal{I}^1_{[a,b]} \; y$$

$$x \; \mathcal{I}_{[a,b]} \; y \quad \Leftrightarrow \quad a \to^{>1} b | x \; \mathcal{I}^{>1}_{[a,b]} \; y$$

**Reuse additional constraints**

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

- 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

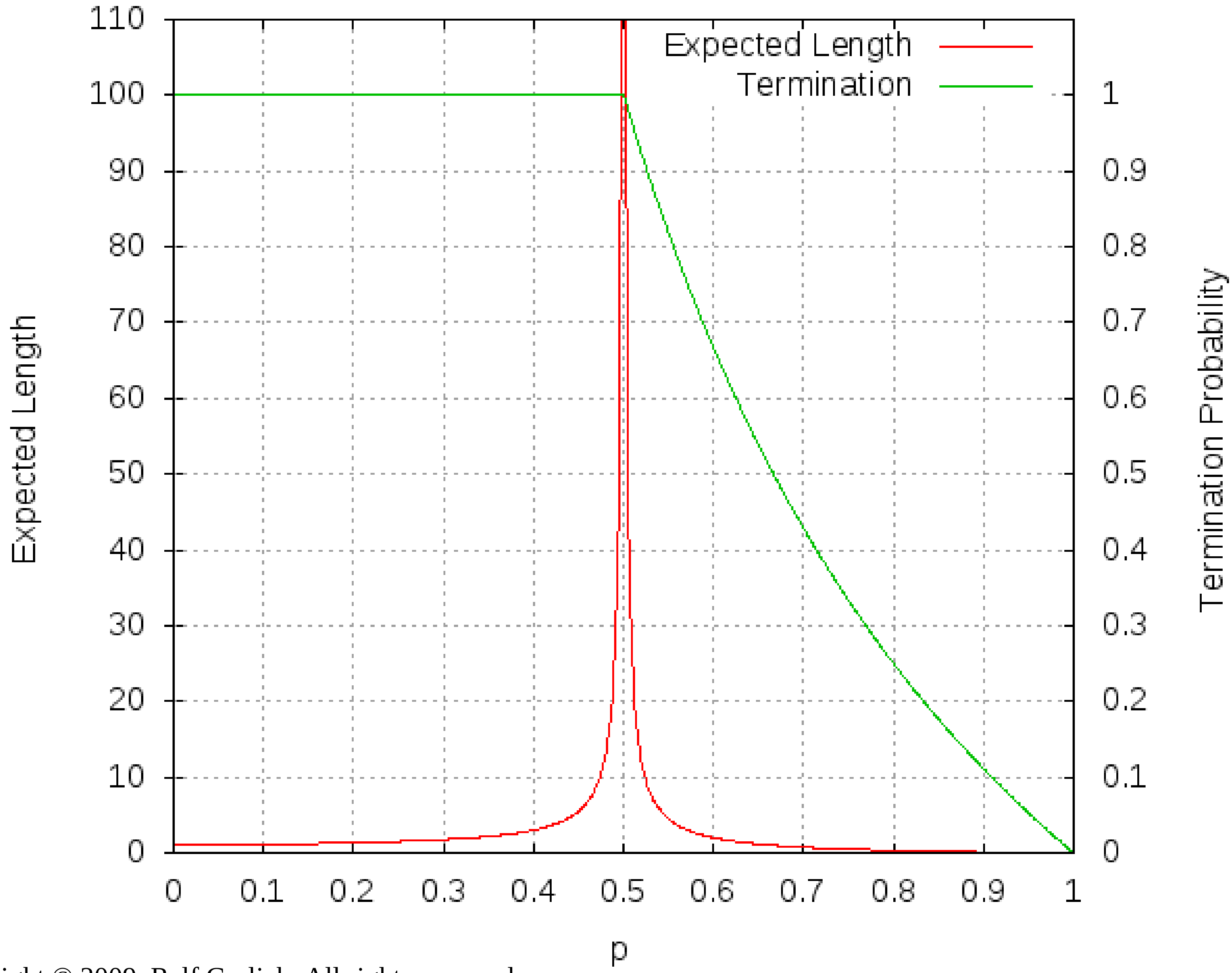
Copyright © 2009, Ralf Gerlich. All rights reserved.

# 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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Example Distribution




Copyright © 2009, Ralf Gerlich. All rights reserved.

# Implementation Considerations

- Buitin-Solver
  - FD-Solver is not good at detecting contradictions
  - e.g. $a<b \wedge b<a$ leads to long step-by-step domain reduction
  - Solution: Combine Axioms with FD-rules
  - Implemented in SWI CHR (11 constraints, 80 rules)
  - e.g. $a<b \wedge b<a \Rightarrow \bot$.
  - e.g. $a+b=c \backslash a+b=d \Leftrightarrow c=d$.
- Emulation of Memory-Access
  - Pointers and Arrays of arbitrary depth (e.g. C, C++)
  - `read(State,Ref,Var)`
  - `write(NewState,OldState,Ref,Var)`
  - Requires efficient lookup in constraint store

Copyright © 2009, Ralf Gerlich. All rights reserved.

# The K.U. Leuven Compiler System

- Optimising compiler
  - Elimination of passive / dead rules
  - Compression of sucessive 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

Copyright © 2009, Ralf Gerlich. All rights reserved.

# 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, $CHR^{v}$, refined semantics)
  - Optimisation potential
  - Static Checking
  - Search in non-Prolog-based environments
- Non-Priorities
  - Status as general-purpose language

Copyright © 2009, Ralf Gerlich. All rights reserved.

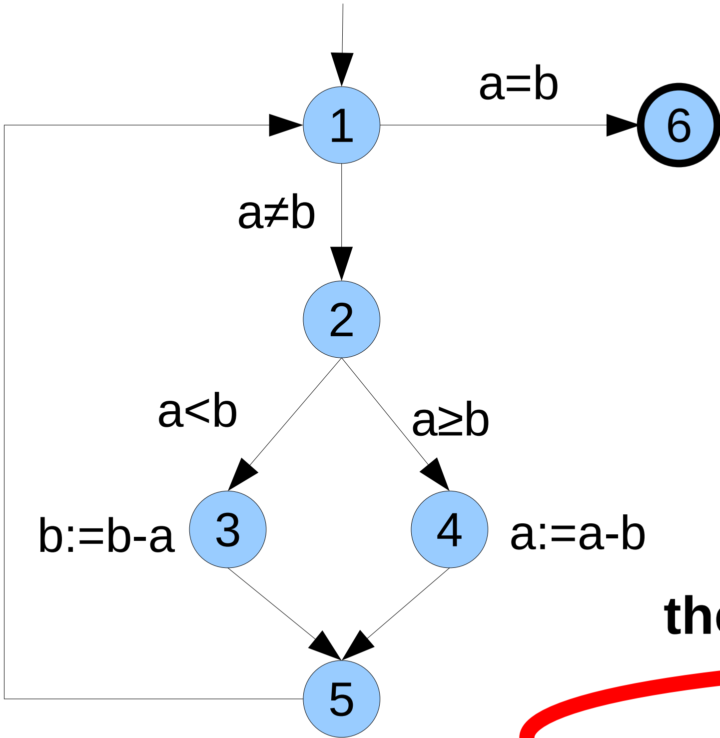
# Questions?

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Backup

Copyright © 2009, Ralf Gerlich. All rights reserved.

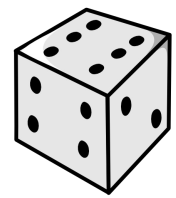
# Denise et al., 2004



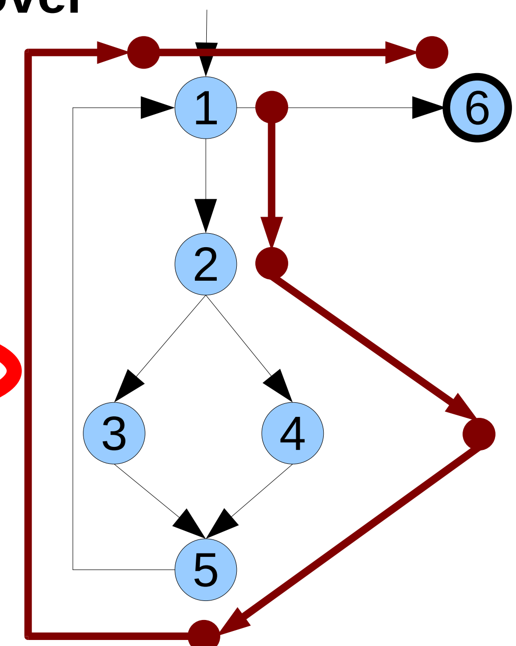

Number of paths of length l starting at node k

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

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



**Randomly select a path (uniform distribution over paths of length L≤N)**




**then construct path constraint**

**No solution? Next try!**

**Large share of infeasible paths makes algorithm impracticable!**

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Gotlieb et al, 1998

```
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),
            if(A2>B2,
                (A3=A2-B2,B3=B2),
                (B3=B2-A2,A3=A2))
```

**Loop unrolling**
**⇒**
**First and further iterations can be handled in parallel**
**⇒**
**“Prediction” of infeasible paths**

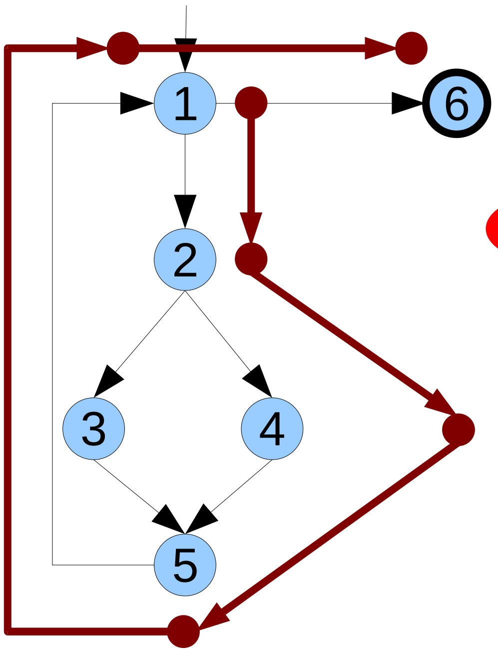
```
Cin ⇒ (while(C,Vin,V,Vout,B) →
              Bin,
              while(C,Vin',V,Vout,B'))
```

mit

```
Bout  =subst(B,Vout→Vin')
Cin   =subst(C,V→Vin)
B'    =subst(B,Vin→Vin')
```

Copyright © 2009, Ralf Gerlich. All rights reserved.

# Nguyen Tran and Deville, 2001




**Input: Control Flow Graph**

**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**

Copyright © 2009, Ralf Gerlich. All rights reserved.

# CHR

**„CHR [Constraint Handling Rules] are essentially a <u>committed-choice</u> language consisting of <u>multi-headed</u> <u>guarded</u> rules that <u>rewrite</u> constraints into simpler ones until they are <u>solved</u>.“**

***Thom Frühwirth, 1994***

| | | | | |
|---|---|---|---|---|
| Simplification: | H | $\Leftrightarrow$ | G | \| C. |
| Propagation: | H | $\Rightarrow$ | G | \| C. |
| Simpagation: | H1\H2 | $\Leftrightarrow$ | G | \| C. |

| | |
|---|---|
| H | User-defined Constraints |
| G | Builtin Constraints |
| C | User-defined+Builtin Constraints |

**Operationelle Semantik: Regeln → Termersetzungen**
**Deklarative Semantik: Regeln → Äquivalenzaussagen**

Copyright © 2009, Ralf Gerlich. All rights reserved.