Using Constraint Satisfaction Formulation and Solution Techniques for Random Test Program Generation

Roy Emek

12-Sep-2002

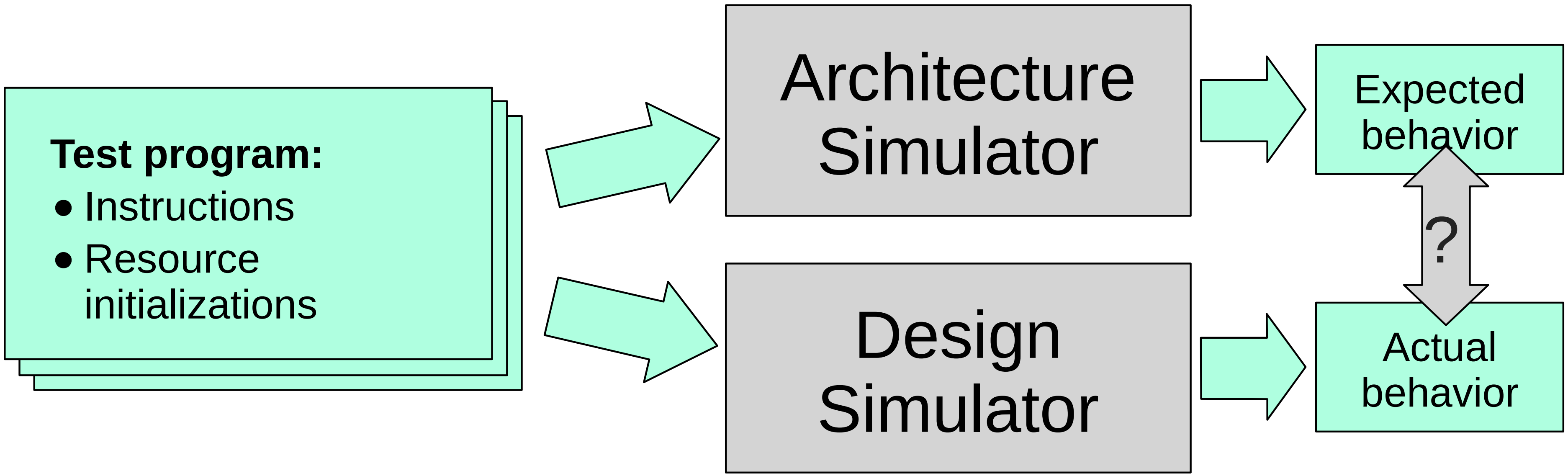
IBM Research Lab in Haifa

Outline

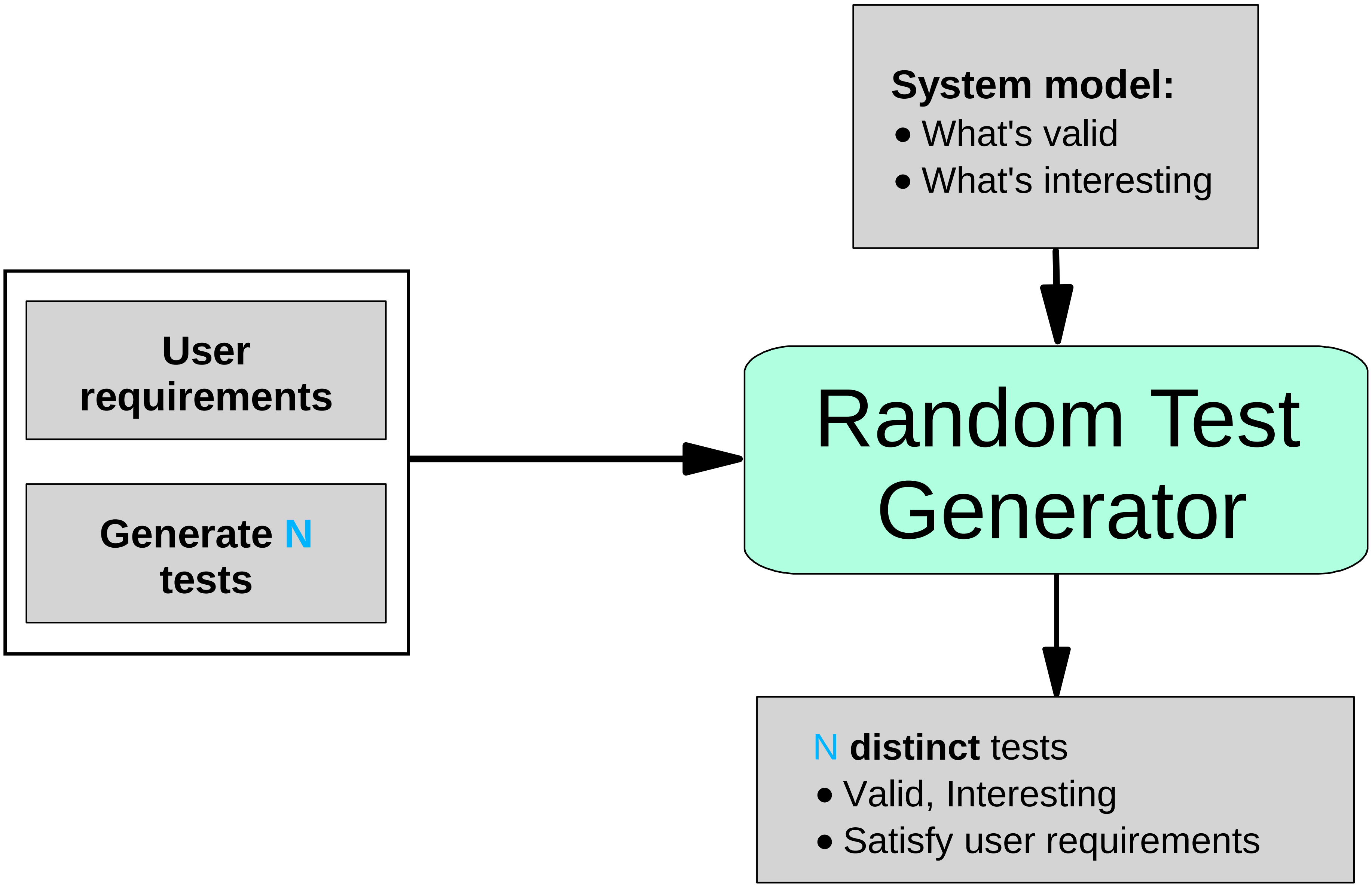
- **Random test program generation**
- **Constraint Satisfaction Problems (CSP)**
- **Modeling test programs as CSP**
- **CSP for random test generation: characteristics**
- **Solution building blocks**

[Based on a paper by E. Bin,
R. Emek, G. Shurek and A. Ziv]

Verification Through Simulation



Random Test Program Generator



Test Program Constraints

Test quality: sum overflow

```
add R1 ← R2 + R3
load Rx ← 1000(Ry)
???? ??, Rz
mult Rz ← R6 × R7
```

Validity: $x \neq y$

User request: same register

CSP Definition

[Mackworth, Freuder, Montanari, Dechter, Rossi, ...]

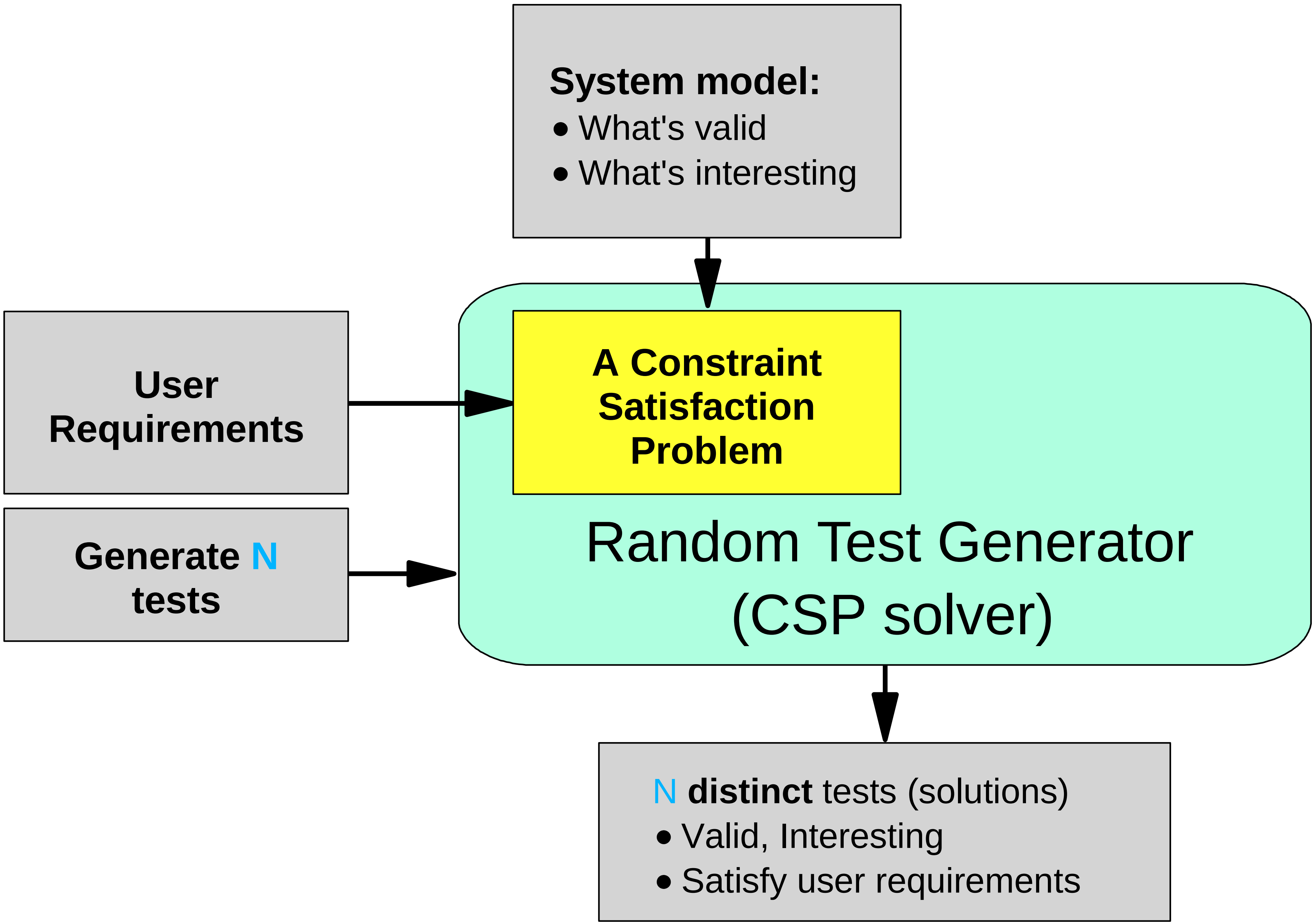
- **Variables of the problem**
  - address, register_value
- **Domain (set) for each variable**
  - address: 0x0000 - 0xFFFF
  - number of bytes in a 'load': { 1, 2, 4, 8, 16 }
- **Constraints (relations) over variables**
  - (load n bytes) $\Rightarrow$ (align address to n bytes boundary)
  - value(base_reg) + displacement = address

CSP Definition (cont)

- **Solution for a CSP**
  - Every variable is assigned a value from its domain
  - The assignments satisfy all the constraints

- **Example**
  - Variables: a, b, c
  - Domains:
    - A = {1,2,3} ; B = {2,3,4,5} ; C = {1,3,5}
  - Constraints:
    - $a^2 < b$; $c \neq b$; $a < c - 1$
  - Solution:
    - $a = 1$; $b = 4$; $c = 3$

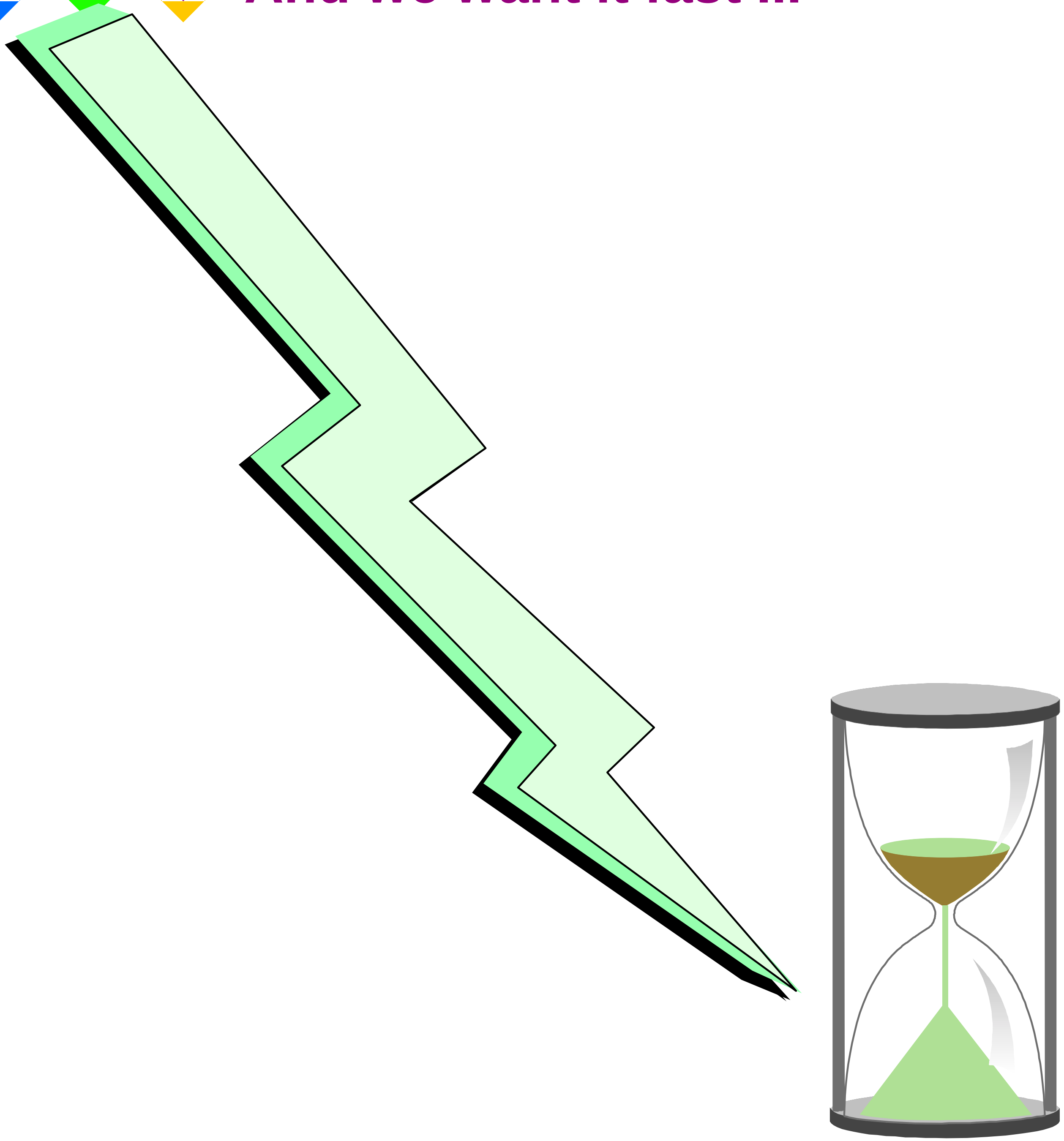
Random Test Program Generator (2)



CSP@RTG Characteristics

- **Random, uniform distribution solution**
  [Yuan et al. '99, Dechter et al. '02]
  - As opposed to one, all, or 'best' solution
- **Huge domains: 2^{64} and more**
  - Example: address space
  - Representing and operating on large sets becomes an issue
- **Hierarchy of constraints** [Borning et al., '87]
  - Mandatory: test case validity
  - Non-mandatory: makes the test 'interesting'

And we want it fast ...



Solution Algorithm:
Consistency - A Single Constraint

$$X \qquad\qquad Y \qquad\qquad Z$$

$$\{1, 2, 3\} \quad \{1, 2, 3\} \quad \{1, 2, 3\}$$

$$R: (x,y,z) \in X \times Y \times Z,\ x=y+z$$

$$\Downarrow$$

$$\{\cancel{1}, 2, 3\} \quad \{1, 2, \cancel{3}\} \quad \{1, 2, \cancel{3}\}$$

Solution Algorithm: Maintaining Arc Consistency (MAC)

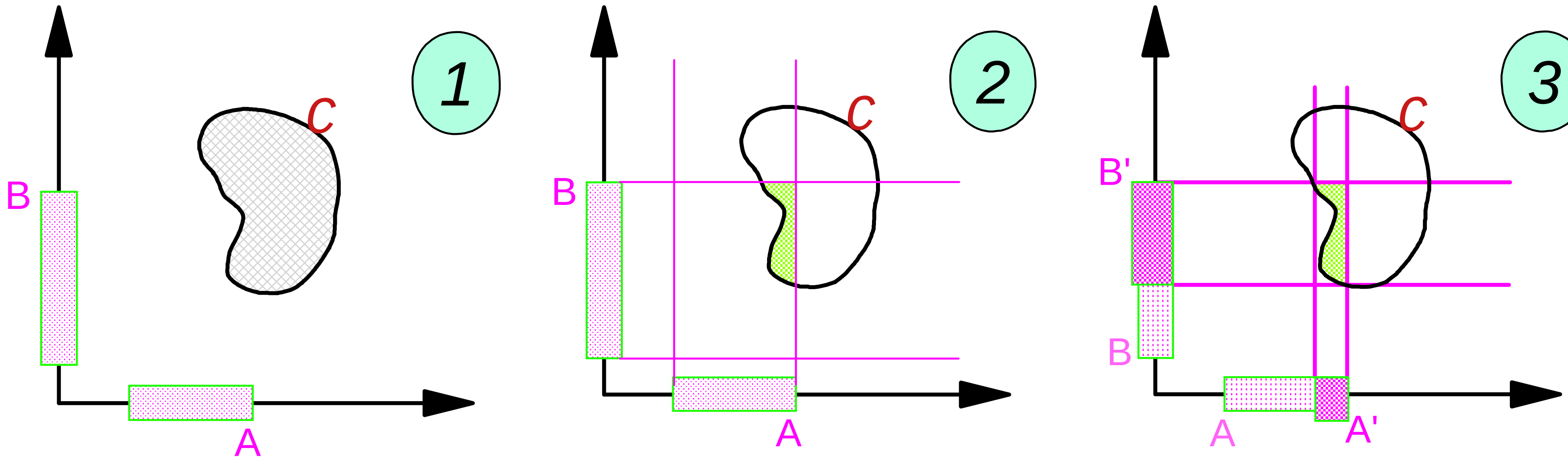
[Mackworth, 1977]

Arc = Constraint

The process: reducing domains to single-values

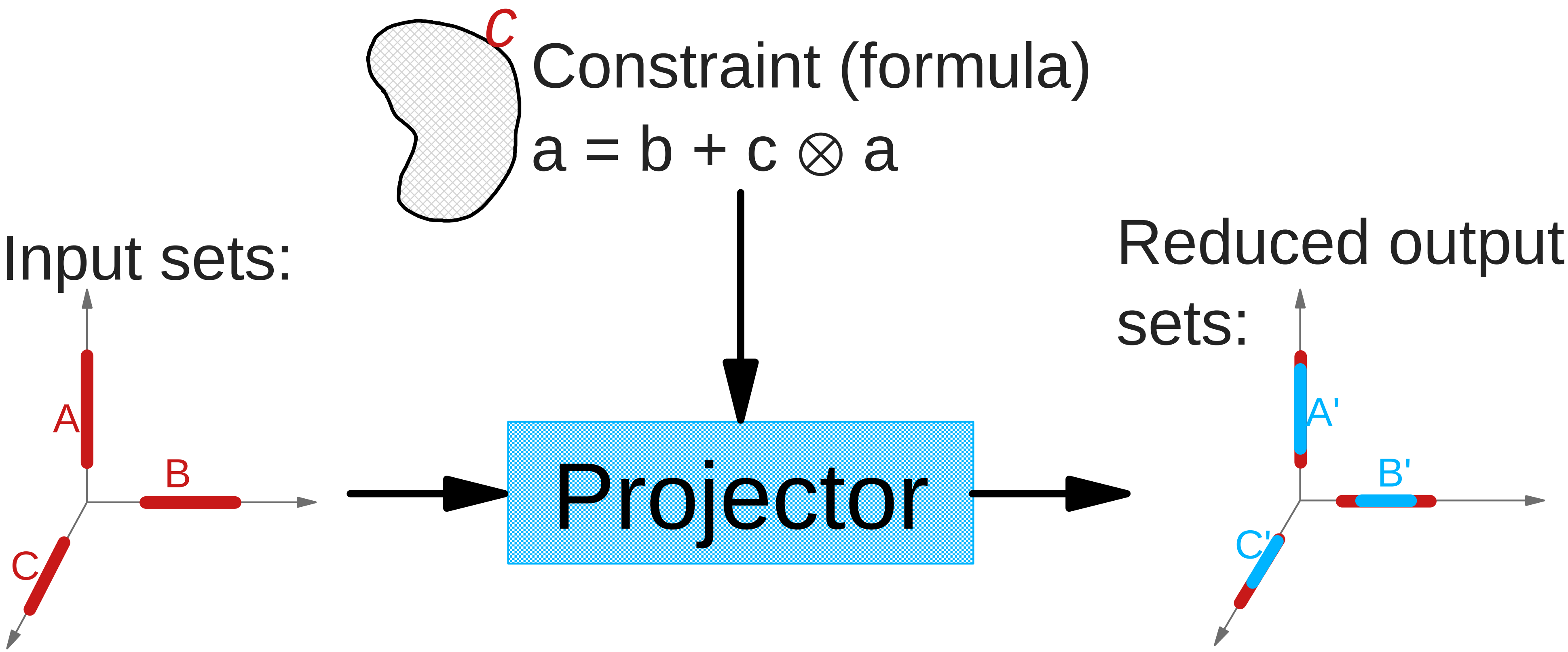
1. **Make all constraints locally consistent**
   - Some constraints are handled repeatedly
   - Achieve fixed-point
2. **Choose a variable:** $address$
3. **Choose a value:** $address \leftarrow 0x1234$
   - $0x1234 \in domain\ (\ address\ )$
4. **Go to step 1**
5. **On failure - backtrack**
   - Failure results in an empty set / domain

Consistency as Projection



Formula Based Constraint Projector

- **MAC scheme – projectors for constraints**
- **Developing arithmetic / logical / bit-wise projectors time after time again ?**
  - a=b+c, a+b=c+d, a + b = c *bit-xor* d, ...
  - Error prone, labour intensive



Example: Projecting Disjoint Constraints

Constraint: $(a=b) \vee (b=a+c) \vee (c=3 \cdot a)$
Domains: A={1,2,3}; B={3,4,5};C={4,5}

(1) Project sub-constraints separately

(2) Join* sub-constraints projections

	A	B	C
Input domain	{1,2,3}	{3,4,5}	{4,5}
a=b	{3}	{3}	-
b=a+c	{1}	{5}	{4}
c=3a	ϕ	-	ϕ
Results	{1,3}	{3,5}	{4,5}

Domain (Set) Representation Example: bit-vectors

- **Origin of the challenge: large H/W resources**
  - 128-bit registers
  - 64-bit wide memory address space
- **All the addresses such that**
  - addr = base + displacement // architectural
  - addr[3:6] = 01?1 // cache line
  - addr $\in$ [0x2000 : 0x10FFF] // memory space
- **'Masks' (DNF) representation:**
  - 01?1 $\rightarrow$ 0101, 0111

Problem: Exponential Explosion

- **01010101 + 0?0?0?0? →**
  - {10101010, 01101010, 10011010, 01011010, 10100110, 01100110, ..., 10010101}
- **The general case: a + b → $2^{(n/2)}$ clauses**
- **Coping with the problem**
  - Binary Decision Diagrams (BDDs)
    - Sometimes: space explosion
  - Approximations

We only have partial solutions

Summary

- **Viewing test generation as CSP**
- **Characteristics: random, huge domains**
- **Solution scheme**
  - Consistency based

- **HRL's test generators are CSP based**
  - The basis for test generators for all the processors designed in IBM

Using Constraint Satisfaction Formulation and Solution Techniques for Random Test Program Generation

End of Presentation

12-Sep-02

IBM Research Lab in Haifa

Floating Point Operations – Stochastic Approach

- **a·2$^{\alpha}$ op b·2$^{\beta}$ = c·2$^{\gamma}$**
  - op: +, -, ·, ...
  - Limited number of bits:
    non-continuous domain, rounding
- **Constraints:**
  - 'op' itself
  - bit #n = '0'
  - Number of '1's = m
  - a $\in$ [a_1 ... a_2]

mantissa:53	exp:11
mantissa:53	exp:11

op

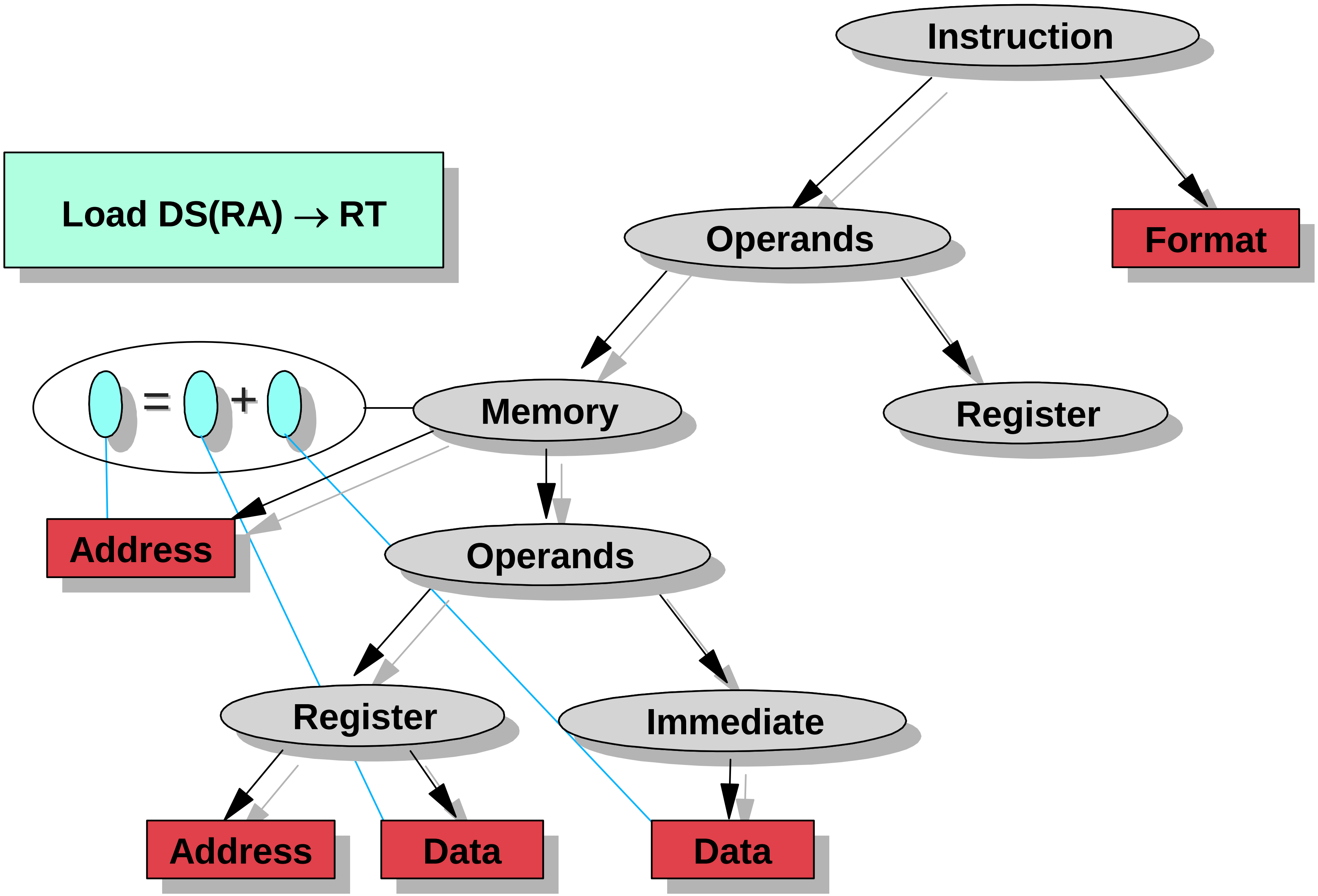
mantissa:53	exp:11

Stochastic solution scheme:

- **assign random 64 $\times$ 3 bits**
- **Hill-climbing**
  - Simple heuristics
  - Local maximum: flip random bits
  - After some time - give up and start all over again

Instruction Constraints

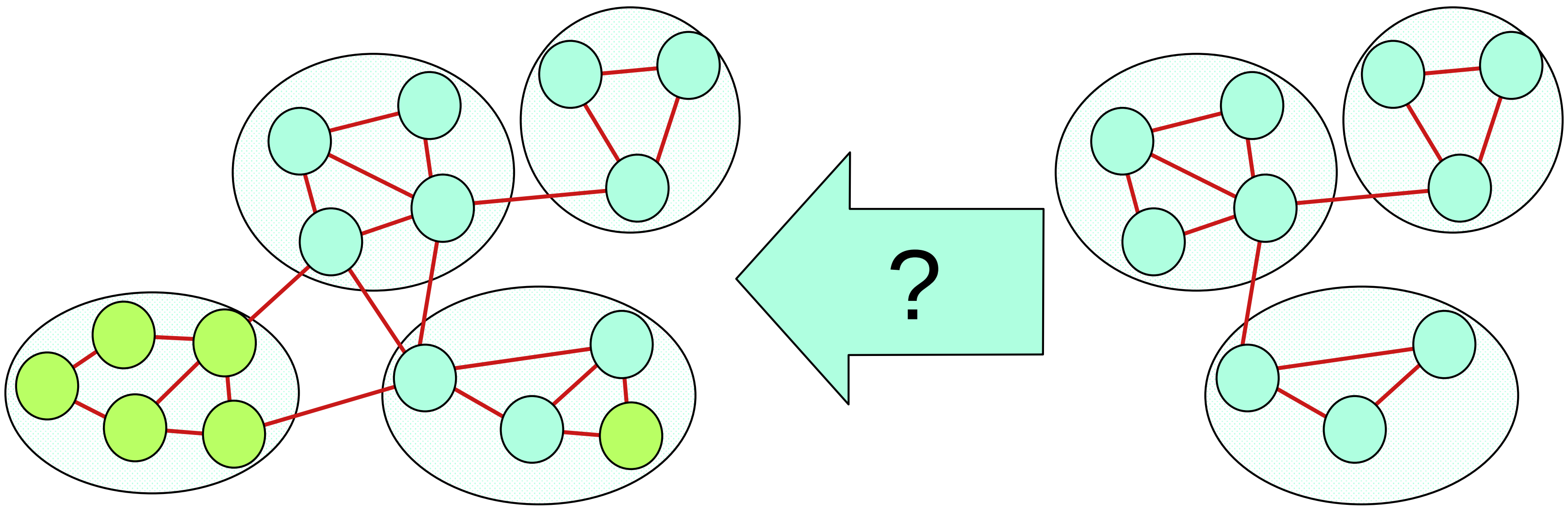
Modelling a PowerPC instruction



Problem Partition, Dynamic Modeling

- **Partitioning the problem**
  - Easier to model, easier to solve
  - Hard to handle interdependencies

- **Dynamic problem structure**



CSP Definition

[Mackworth, Freuder, Montanari, Dechter, Rossi, ...]

- **Variables of the problem**
  - address, register_value
- **Domain (set) for each variable**
  - address: 0x0000 - 0xFFFF
  - number of bytes in a 'load': { 1, 2, 4, 8, 16 }
- **Constraints (relations) over variables**
  - (load n bytes) $\Rightarrow$ (align address to n bytes boundary)
  - value(base_reg) + displacement = address
  - last_instruction = "branch" ?
    - yes: PC = branch-target
    - no : PC = increase (last_instruction_address)