

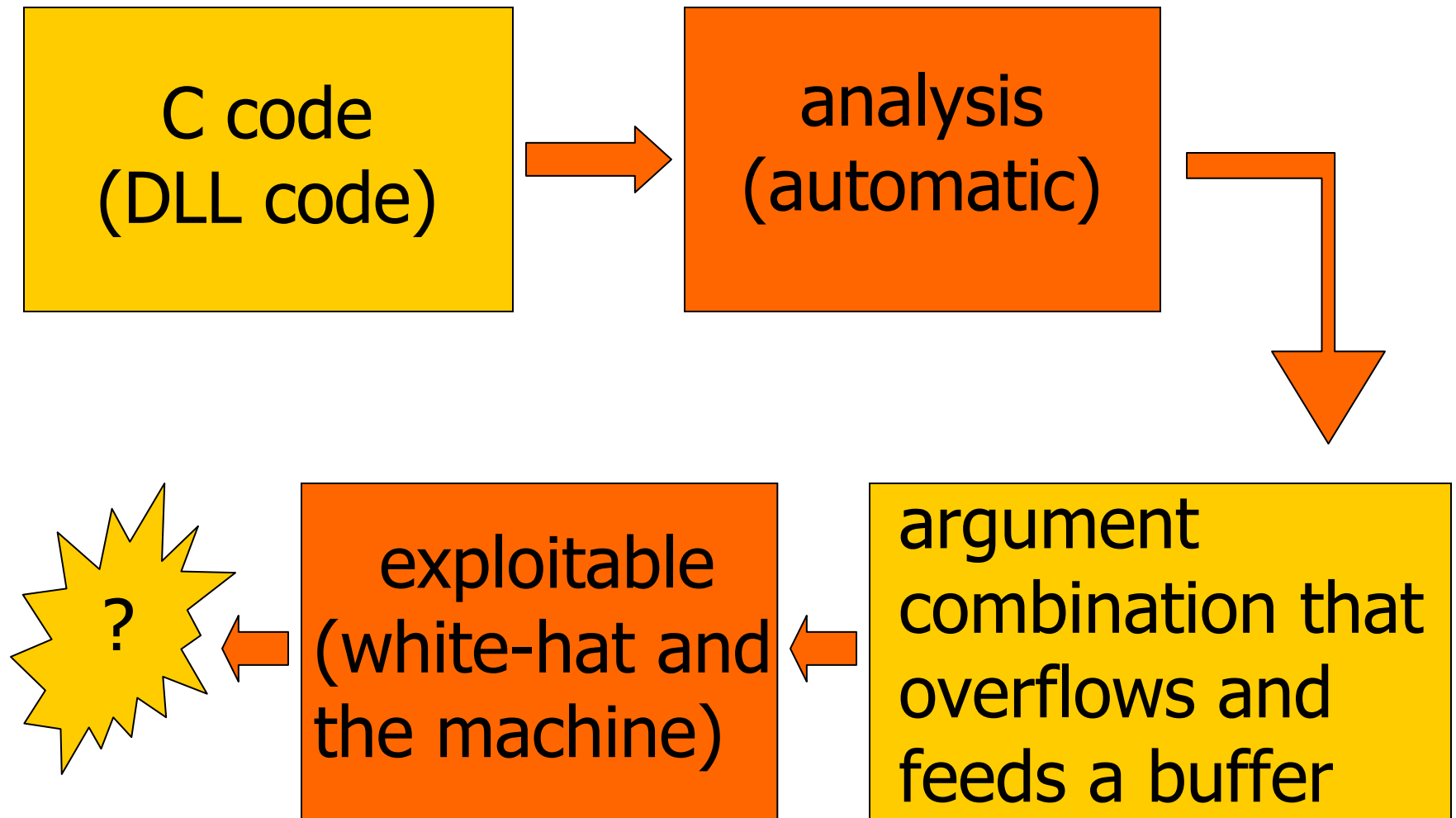
On Modular-Reduction Vulnerabilities

Andy King
University of Kent
a.m.king@kent.ac.uk



Portcullis Computer Security
mu-b@65535.com and doc@65545.com

White-hat agenda



deattack.c of OpenSSH version 2.3.0 (1 of 2)

```
#define SSH_MAXBLOCKS    (32 * 1024)
#define SSH_BLOCKSIZE    (8)
...
#define HASH_MINSIZE     (8 * 1024)
#define HASH_ENTRYSIZE   (2)
#define HASH_FACTOR(x)   ((x) * 3/2) ←remember
...
int ← 87387 ≤ len ≤ 262144 and len is a multiple of 8
detect_attack(unsigned char *buf, u_int32_t len,
              unsigned char *IV)
{
    static u_int16_t *h = (u_int16_t *) NULL;
    static u_int16_t n = HASH_MINSIZE / HASH_ENTRYSIZE;
    register u_int32_t i, j;      ↑n = (8 * 1024) / 2 = 4096
    u_int32_t l;
    ...
}
```

deattack.c of OpenSSH version 2.3.0 (2 of 2)

```

u_int32_t l;
...
↓ len ≤ 262144 = (32 * 1024) * 8
if (len > (SSH_MAXBLOCKS * SSH_BLOCKSIZE) ||
    len % SSH_BLOCKSIZE != 0) ← len % 8 = 0
{ fatal("detect_attack: bad length %d", len); }
for (l = n; l < HASH_FACTOR(len / SSH_BLOCKSIZE); l = l << 2)
    ; ↑ l = 4096 = 212    ↑ 214 + 1 = 3(87387/8)/2 ≤ 3(len/8)/2
if (h == NULL) {
    debug("Installing crc compensation attack detector.");
    n = l;    ← n = l = 216 or n = l = 218 or ...
    h = (u_int16_t *) xmalloc(n * HASH_ENTRYSIZE);
} else {
    ↑ 0 * 2 = 0
    if (l > n) {
        n = l;
        h = (u_int16_t *) xrealloc(h, n * HASH_ENTRYSIZE);
    }
    ↑ 0 * 2 = 0
}
}
}

```

OpenSSH is the “first-fruits” of a new class of vulnerability

- ⌘ So when $87387 \leq \text{len} \leq 262144$ and len is a multiple of 8 then, due to 16 bit overflow, a buffer overflow occurs
- ⌘ Similar vulnerabilities now known to exist in
 - ⊞ glibc 2.1.3
 - ⊞ OpenBSD select()
 - ⊞ Apache chunked encoding memcpy()
- ⌘ But integer overflows are silent and ubiquitous; these are the *known* ones

Existing approaches



- ⌘ Decision procedures such as Fourier-Motzkin:
 - ☒ assume arbitrary precision
 - ☒ do not support logical and bit-ops like `||` and `<<`
- ⌘ Congruence domains [Granger, SAS, 1997; Müller-Olm, ESOP, 2005]
 - ☒ do not support bit-ops
- ⌘ Automatic test data generation [Offutt, SPE, 1999; Gotlieb, CL, 2000]:
 - ☒ model word operations with finite domain constraints
 - ☒ use reification to *enforce overflow somewhere*
 - ☒ solve system and then project onto arguments
 - ☒ need 16- and 32-bit objects but SICStus imposes a painful 30-bit restriction

Tying down a taint (1 of 3)

$$\langle x = \langle b_1, \dots, b_w \rangle, \sigma \rangle \rightarrow \langle \text{skip}, \sigma[\langle b_1, \dots, b_w, 0 \rangle / x] \rangle$$

$$\forall_{i=1}^w \sigma(x)^i = 1$$

$$\langle \text{if } x \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_1, \sigma \rangle$$

$$\forall_{i=1}^w \sigma(x)^i = 0$$

$$\langle \text{if } x \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_2, \sigma \rangle$$

Tying down a taint (2 of 3)



$$b_i = \sigma(y)^i \wedge \sigma(z)^i \text{ for all } i \in [1, w]$$
$$b_{w+1} = \sigma(y)^{w+1} \vee \sigma(z)^{w+1}$$

$$\langle x = y \ \& \ z, \ \sigma \rangle \rightarrow \langle \text{skip}, \ \sigma[\langle b_1, \dots, b_w, b_{w+1} \rangle / x] \rangle$$

Tying down a taint (3 of 3)

$$c_1 = 0$$

$$c_{i+1} = (\sigma(y)^i \wedge \sigma(z)^i) \vee (\sigma(y)^i \wedge c_i) \vee (\sigma(z)^i \wedge c_i)$$

for all $i \in [1, w]$

$$b_i = \sigma(y)^i \oplus \sigma(z)^i \oplus c_i \quad \text{for all } i \in [1, w]$$

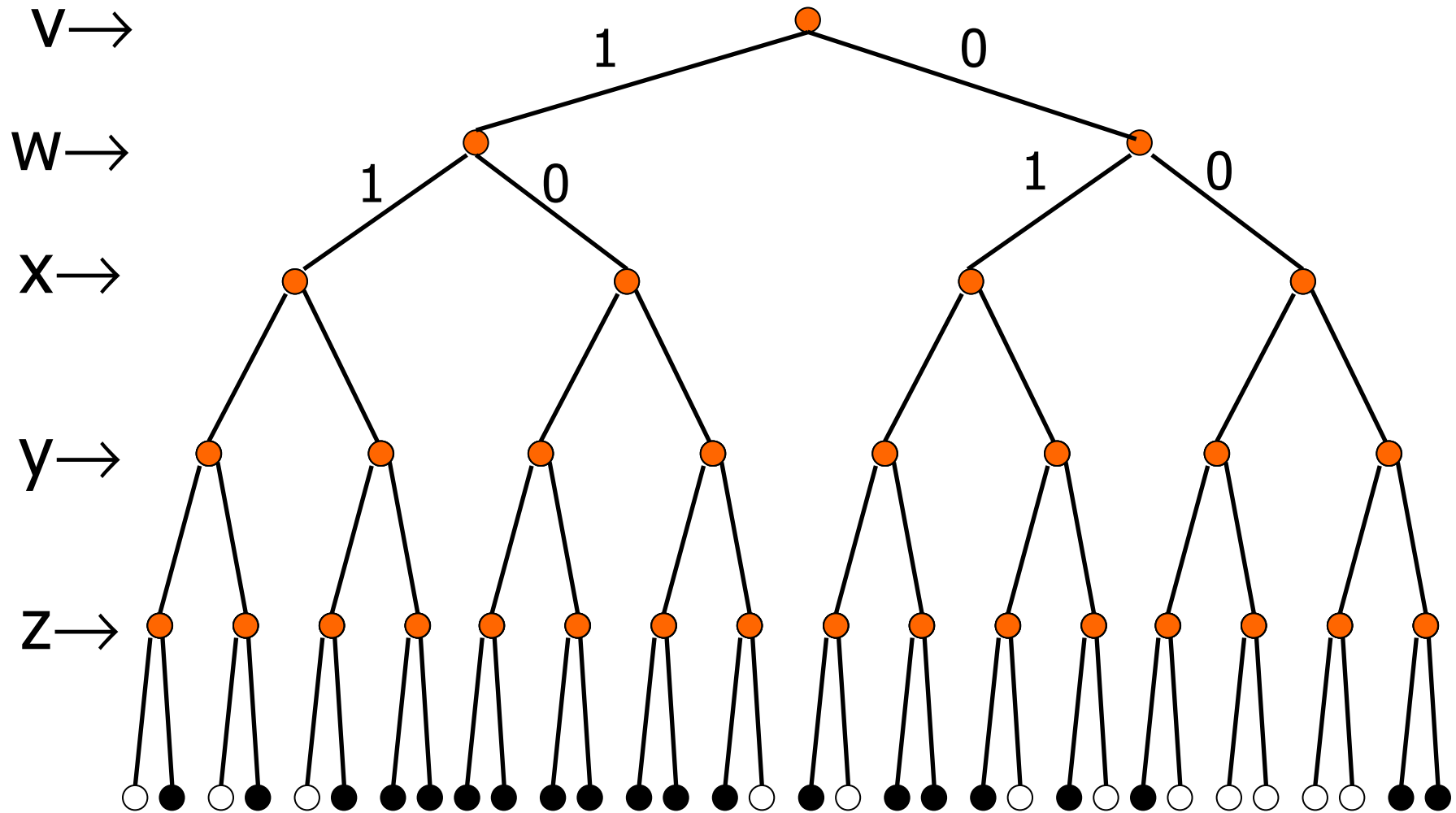
$$b_{w+1} = \sigma(y)^{w+1} \vee \sigma(z)^{w+1} \vee c_{w+1}$$

$$\langle x = y + z, \sigma \rangle \rightarrow \langle \text{skip}, \sigma[\langle b_1, \dots, b_w, b_{w+1} \rangle / x] \rangle$$

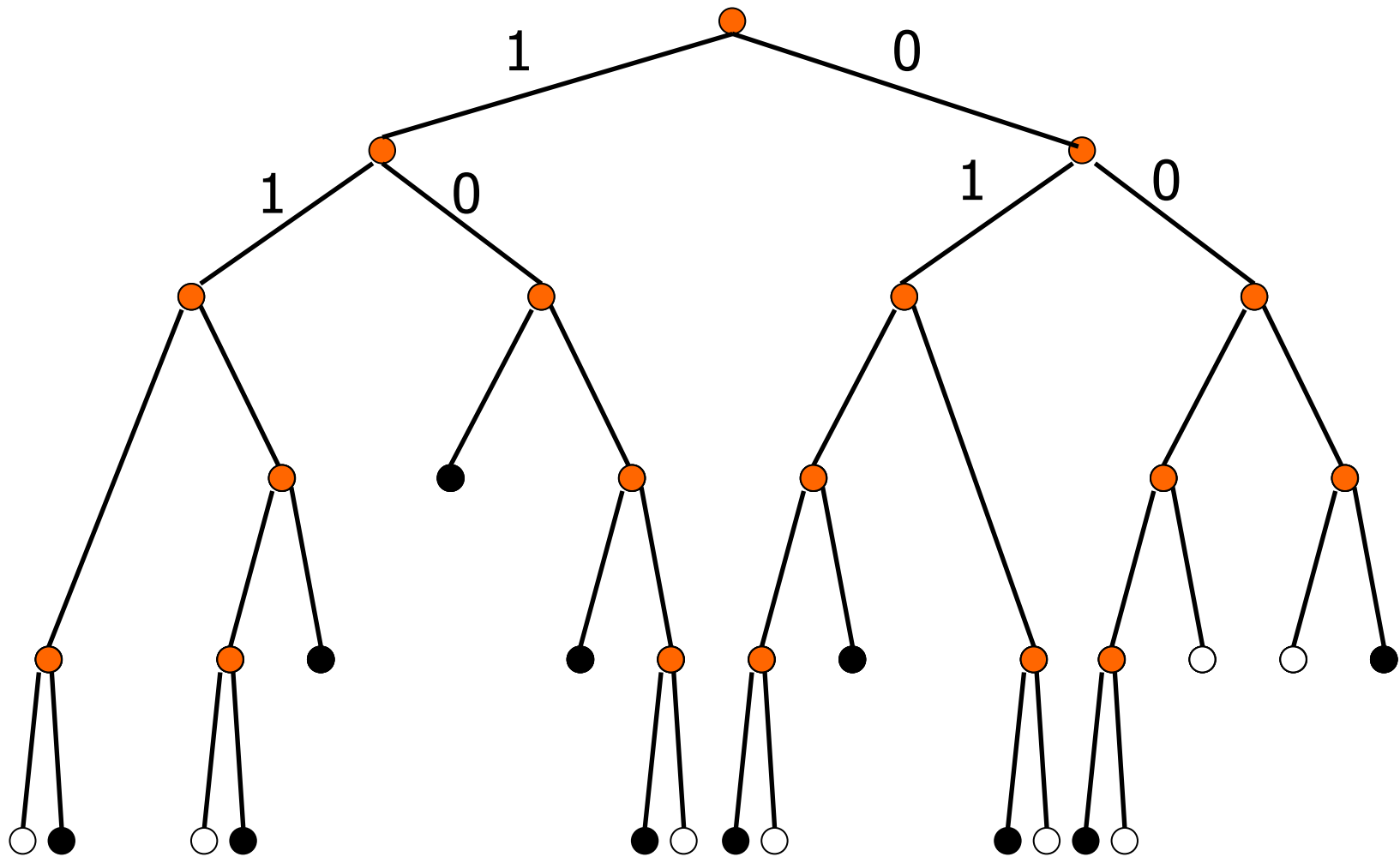
Formalising a hack

- ⌘ Rules for ==, !=, || and && can be defined with
 - ☑ rules for <=, | and &
- ⌘ Let $C \rightarrow^k C'$ iff $C' = \{c' \mid c \in C \text{ and } c \rightarrow^k c'\}$
- ⌘ Then $\text{hack}^k(s) = \{\sigma \mid \dots\}$
 - ☑ $\{\langle s, \sigma \rangle\} \rightarrow^k \{\langle (x = \text{malloc}(y); s'), \sigma' \rangle\}$ and
 - ☑ $\sigma'(y)^{w+1} = \text{true}$
- ⌘ With $\sigma \in \text{hack}^k(s)$, s will *surely* encounter an overflow fed malloc at step k
- ⌘ Note that $\sigma \in \text{hack}^{42}(s)$ is simpler to check and, thus exploit, than $\sigma' \in \text{hack}^{123}(s)$

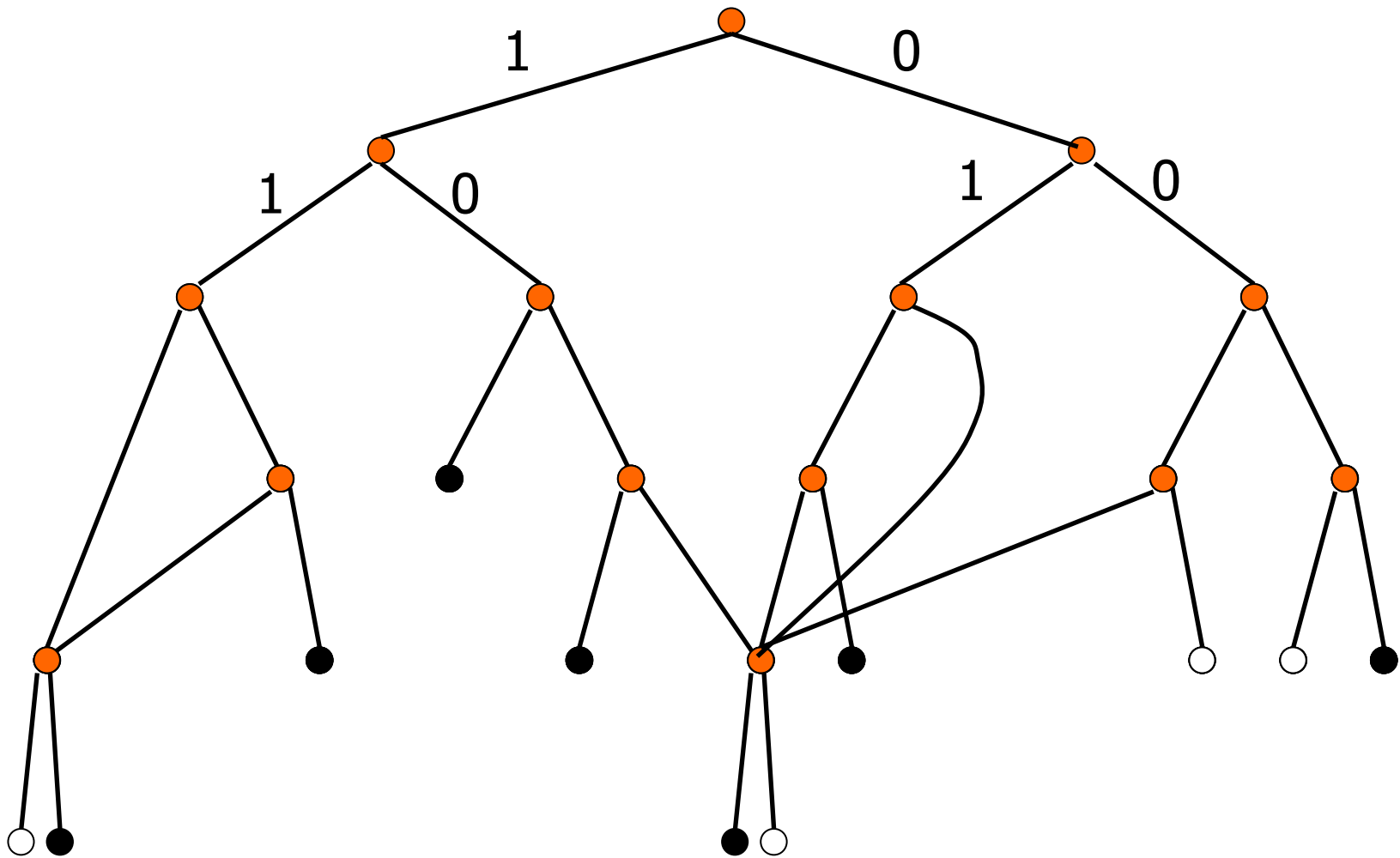
OBDD: an API on bit-operations



ROBDD (reduction 2)



ROBDD (reduction 3)



Encoding taint semantics in ROBDDs [Hawkins, JAIR, 2005]

⌘ Use function f to encode behaviour of s by:

⊠ if $\langle s, \sigma \rangle \rightarrow \langle \text{skip}, \sigma' \rangle$ then

⊠ $f = \alpha(\sigma) \wedge \alpha'(\sigma')$ where

⊠ $\alpha(\sigma) = \bigwedge_{x \in \text{var}(s)} \bigwedge_{i \in [1, w+1]} x_i \leftrightarrow \sigma(x)^i$

⊠ $\alpha'(\sigma') = \bigwedge_{x \in \text{var}(s)} \bigwedge_{i \in [1, w+1]} x'_i \leftrightarrow \sigma'(x)^i$

⌘ If $s = (x = y \ \& \ z)$ then $[[s]] = \alpha(\sigma) \wedge \alpha'(\sigma') =$

⊠ $(\bigwedge_{i \in [1, w]} x'_i \leftrightarrow (y_i \wedge z_i)) \wedge$

⊠ $(x'_{w+1} \leftrightarrow (y_{w+1} \vee z_{w+1})) \wedge$

⊠ $(\bigwedge_{i \in [1, w+1]} y'_i \leftrightarrow y_i) \wedge (\bigwedge_{i \in [1, w+1]} z'_i \leftrightarrow z_i)$

Ensuring a taint

- ⌘ Consider $x = y + z; w = \text{malloc}(x)$
- ⌘ Interested in values of x, y and z which induce an overflow:
 - ⊞ $f \wedge [[x = y + z]] \models x'_{33}$ where f is defined over x_1, \dots, z_{33}
 - ⊞ $f \models ([[x = y + z]] \rightarrow x'_{33})$ where f is defined over x_1, \dots, z_{33}
 - ⊞ $f = \forall_{x'_1} \dots \forall_{z'_{33}} ([[x = y + z]] \rightarrow x'_{33})$ where $\forall_y(g) = g[0/y] \wedge g[1/y]$
- ⌘ Mix of constraint solving, deduction with ...
 - ⊞ If $[[x = y + z]] \models g$ then use g instead of $[[\dots]]$

Need for approximation

⌘ Prototype implementation:

⊡ ROBDD explosion on \ll

⌘ Tactics for tractability:

⊡ Variable reordering [Rudell, ICCAD, 1993]

⊡ Early projection [Vardi, CP, 2001]

⊗ $||\exists_x(f)|| < ||f||$ where $||\cdot||$ counts nodes

⊡ ROBDD approximation [Ravi, DAC, 1998]

⊗ $||f|| > ||g||$ where $f \models g$

⊡ ROBDD approximation [Kettle, TACAS, 2006]

⊗ $||f|| \approx 2||g||$ where $f \models g$



The End