CONSTRAINT HANDLING RULES

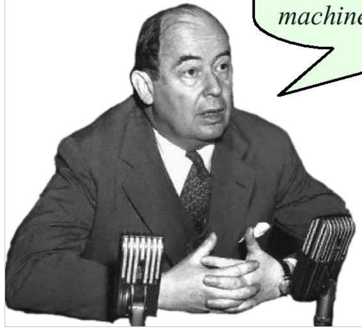
an introductory tutorial

Jon Sneyers
October 2009

---

von Neumann quote

> *"You insist that there is something that a machine can't do. If you will tell me **precisely** what it is that a machine cannot do, then I can always make a machine which will do just that."*



John von Neumann (1903-1957)
Hungarian-American mathematician, pioneer of computer science

---

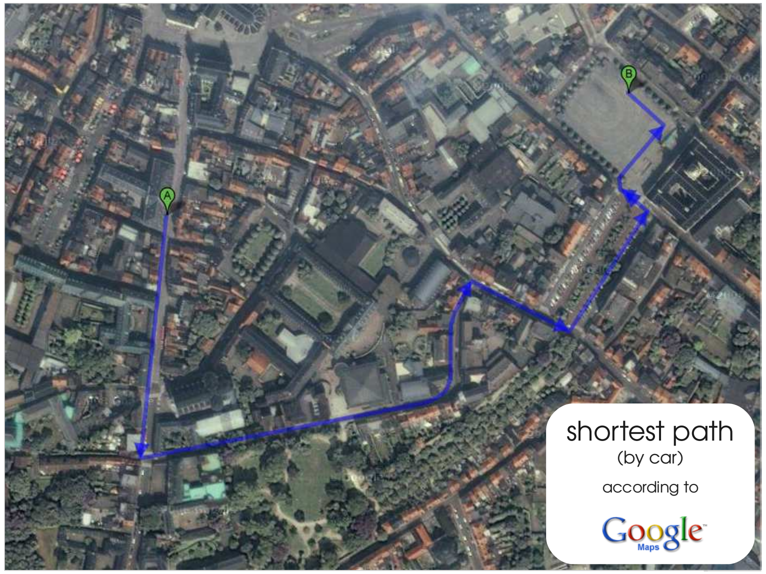
PART ONE

Introduction

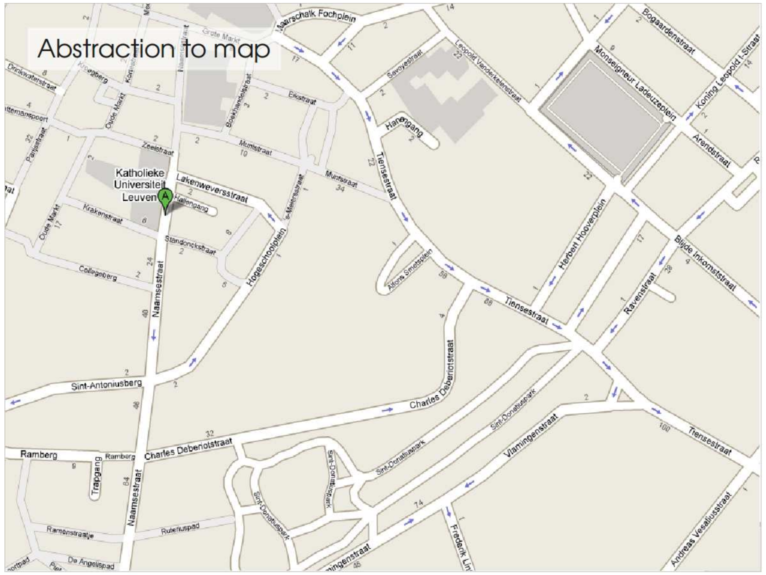
How to get from A to B ?

point B : Library
Ladeuzeplein, Leuven

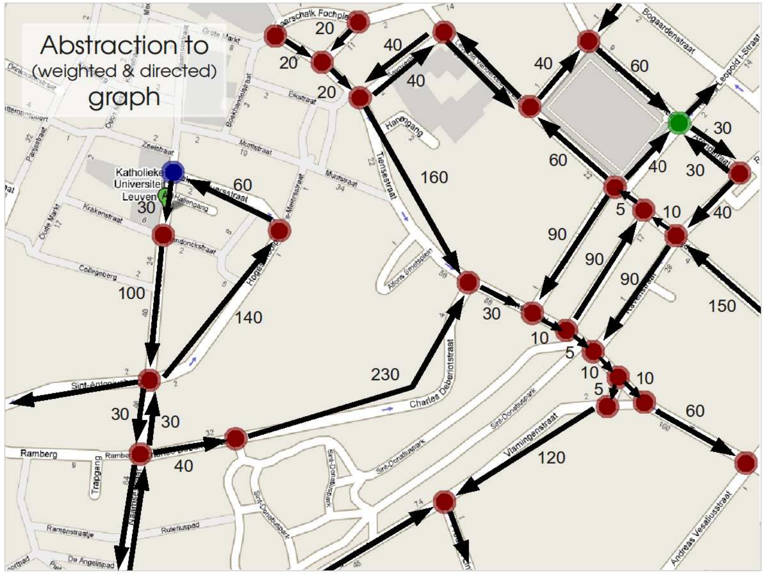
point A : Universiteitshallen
Naamsestraat 22, Leuven, Belgium



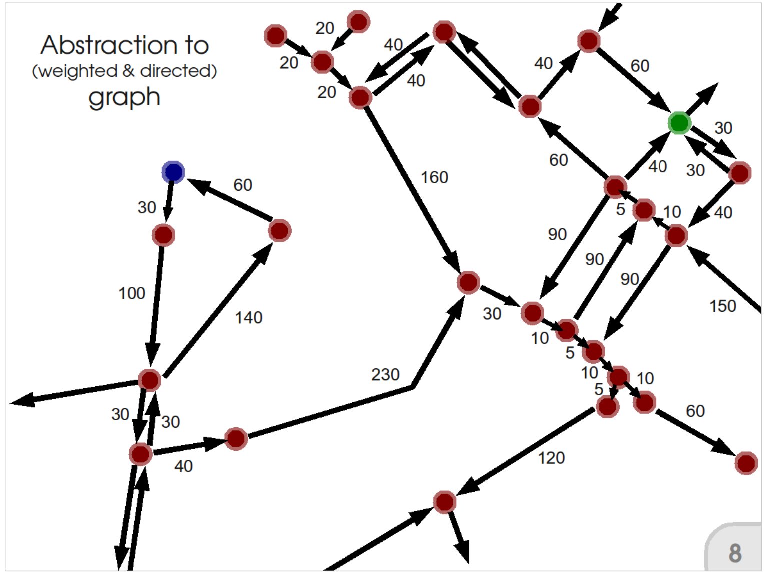
shortest path
(by car)
according to
Google Maps



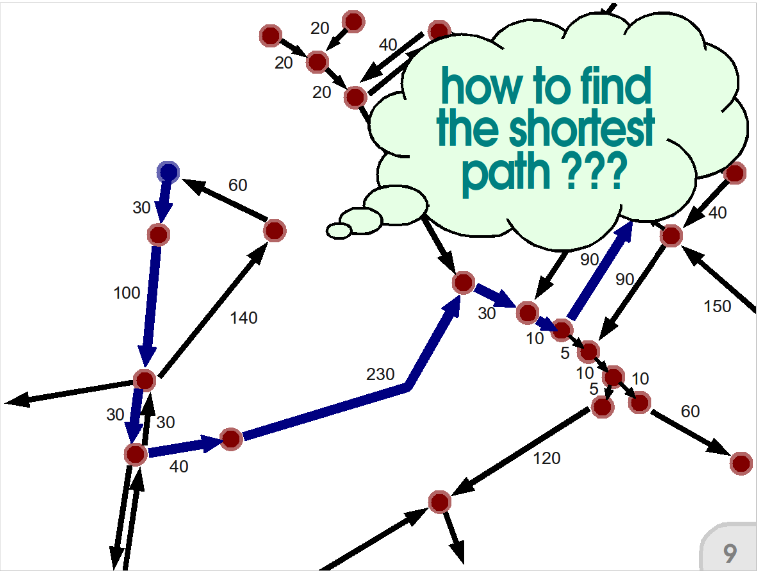
Abstraction to map



Abstraction to
(weighted & directed)
graph



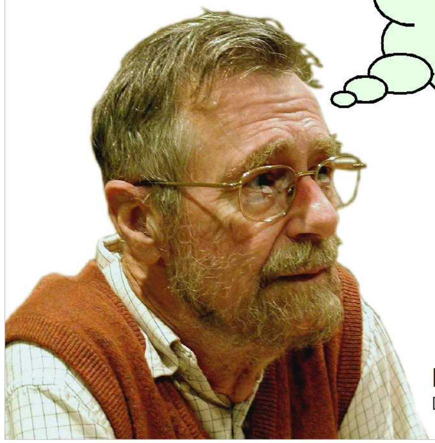
Abstraction to
(weighted & directed)
graph



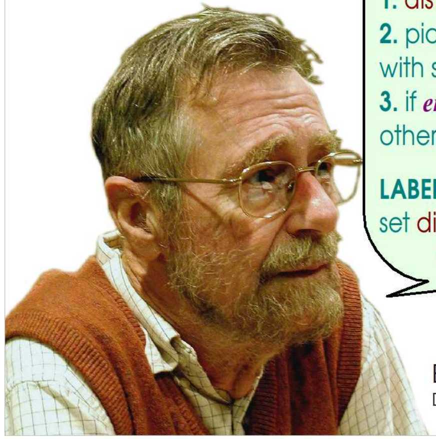
how to find the shortest path ???

---

how to find the shortest path ???



Edsger Dijkstra (1930-2002)
Dutch computer scientist



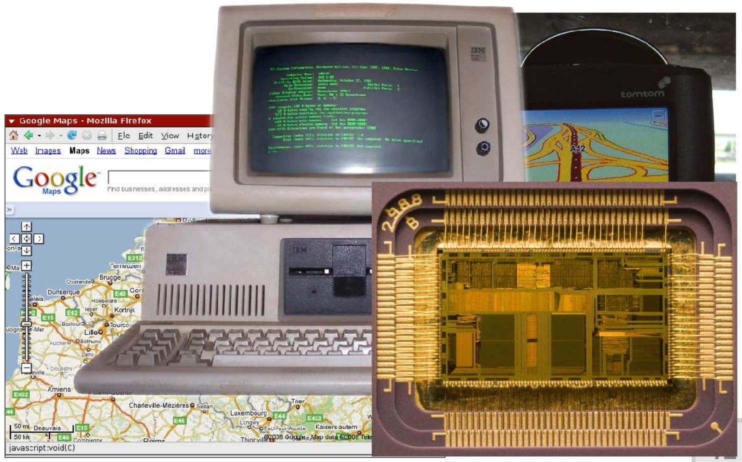
Dijkstra's algorithm:

1. distance(*start-point*) = *0*
2. pick a (not-yet-considered) point *x* with smallest distance, **LABEL(*x*)**
3. if *end-point* is considered, stop; otherwise go to step **2**

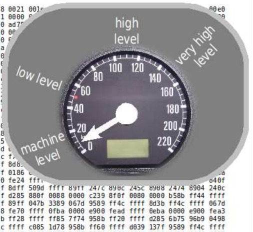
LABEL(*x*): for all arrows $x \xrightarrow{a} y$:
set distance(*y*) = distance(*x*) + *a*
(if the new distance is shorter)

Edsger Dijkstra (1930-2002)
Dutch computer scientist

How to do this **automatically** ?



Implementing Dijkstra's algorithm

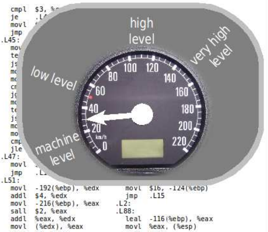


```
457f 464c 0101 0001 0000 0000 0000 0000 0002 0003 0001 0000 84b0 0804 0034 0000 3620 0000 0000 0000 0034 0020 0007 0028 0021                                              00e0
0000 00e0 0000 0005 0000 0004 0000 0003 0000 0114 0000 8114 0804 8114 0804 0013 0000 0013 0000 0004 0000 0001 0000 0001 0000
0005 0000 1000 0000 0001 0000 1d00 0000 ad00 0804 ad00 0804 0194 0000 01e4 0000 0006 0000 1000 0000 0002 0000 1d78 0000
0000 0004 0000 0128 0000 8128 0804 8128 0804 0020 0000 0020 0000 0004 0000 0004 0000 e551 6474 0000 0000 0000 0000 0000
6c2f 2d64 696c 756e 2e78 6f73 322e 0000 0004 0000 0010 0000 0001 0000 4e47 0055 0000 0000 0002 0000 0002 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0001 0000 0002 0000 0003 0000 0006 0000 0004 0000 0007 0000 0005 0000 0008 0000 000a
0000 0000 004e 0000 0000 0000 006d 0000 0012 0000 0049 0000 0000 0000 0171 0000 0012 0000 007b 0000 0000 0000 00fa 0000
0000 0000 0000 0036 0000 0012 0000 0067 0000 0000 0000 00d1 0000 0012 0000 0053 0000 0000 0000 02e0 0000 0012 0000
00b1 0000 0012 0000 005a 0000 ae94 0804 0004 0000 0011 0016 006c 0000 9804 0804 0004 0000 0011 000e 000f 0000 0000
0000 0023 0000 0000 0000 0000 0020 0000 6c00 6269 2e6d 6f73 362e 6c00 676f 5f00 764a 525f 6765 7369 6574 4372
6362 732e 2e6f 0036 7270 6e69 6674 7400 6d69 7365 6700 7465 0073 6566 666f 6300 6c61 6f6c 0063 7473 6964 006e 7373
5f00 6c5f 6269 5f63 7473 7261 5f74 616d 6e69 6600 6572 0065 4c47 4249 5f43 2e32 0030 0000 0002 0002 0002 0002 0002
0001 0000 0010 0000 0020 0000 6910 0d69 0000 0003 0092 0000 0000 0000 0001 0001 0032 0000 0010 0000 0000 0000 6910
0804 0a05 0000 ae68 0804 0107 0000 ae6c 0804 0207 0000 ae70 0804 0307 0000 ae74 0804 0407 0000 ae78 0804 0507 0000
ae88 0804 0907 0000 ae8c 0804 0d07 0000 8955 83e5 08ec e5e8 0000 e800 014c 0000 a7e8 0013 c900 00c3 35ff ae60 0804
ffff 25ff ae6c 0804 0868 0000 e900 ffd0 ffff 25ff ae70 0804 1068 0000 e900 ffc0 ffff 25ff ae74 0804 1868 0000 e900
0804 2868 0000 e900 ff90 ffff 25ff ae80 0804 3068 0000 e900 ff80 ffff 25ff ae84 0804 3868 0000 e900 ff70 ffff 25ff ae88
e900 ff50 ffff 0000 0000 ed31 895e 83e1 f0e4 5450 6852 9740 0804 e068 0496 5108 6856 9220 0804 5be8 ffff f4ff 9090 8955
74c0 ff02 8bd0 fc5d c3c9 9090 9090 9090 9090 9090 8955 83e5 08ec 3d80 ae98 0804 7500 a12d ad08 0804 108b d285 1b74 b68d
75d2 c6eb 9805 04ae 0108 c3c9 f689 8955 83e5 08ec 58a1 04ae 8508 74c0 b821 0000 0000 c085 1874 04c7 5824 04ae e808 7a9c
9090 9090 9090 9090 3155 89d2 57e5 c931 3156 53f6 ec81 011c 0000 db31 9589 ff20 ffff d231 b589 ff2c ffff 9d89 ff28 ffff
0000 8d8d ff78 ffff 0c89 e824 fe4e ffff c085 840f 0412 0000 be0f 7885 ffff ffff 2c85 ffff 83ff 64f8 840f 03f2 0000 8f0f 0186
0000 8900 2454 8b04 951c ad20 0804 b58b ff2c ffff 04c7 c024 049c 8908 245c 890c 2474 e808 fe10 ffff 04c7 0124 0000 e800 fe24                                              840f
0120 0000 958b ff1c ffff 953b ff64 ffff 8d0f 0104 0000 958d ff58 ffff 86be 0498 8d08 788d ffff 89ff 2454 8d10 54bd ffff 8dff 509d ffff 89ff 247c 890c 245c 8908 2474 8904 240c
e7e8 fffd 83ff 03f8 850f 00bc 0000 8d8b ff50 ffff c985 880f 00a4 0000 858b ff60 ffff c139 8f0f 0096 0000 958b ff54 ffff d285 880f 0088 0000 c239 8f0f 0080 0000 b58b ff44 ffff
bd8b ff1c ffff 9d8b ff40 ffff 44ff 048e 348d 8b92 3885 ffff 89ff bb0c 9d8b ff30 ffff 3c8d 3bf0 4c95 ffff 8bff 58b5 ffff 89ff 047b 3389 067d 9589 ff4c ffff 8d3b ff4c ffff 067d
8d89 ff4c ffff 953b ff48 ffff 067e 9589 ff48 ffff 8d3b ff48 ffff 067e 8d89 ff48 ffff 85ff ff1c ffff 8583 ff30 ffff e908 fe70 ffff 0fba 0000 e900 fead ffff 0eba 0000 e900 fea3
ffff 0dba 0000 e900 fe99 ffff 0cba 0000 e900 fe8f ffff f883 0f70 ff84 0000 0f00 9c8f 0000 8300 6ef8 850f fe72 ffff bd8b ff28 ffff ff85 7f74 958b ff20 ffff d285 6b75 96b9 0498
8d08 7885 ffff bbff 0001 0000 9d89 ff20 ffff b58d ff5c ffff 7489 0824 4c89 0424 0489 e824 fcac ffff c085 317e 858b ff5c ffff c085 1d78 958b ff60 ffff d039 137f 9589 ff4c ffff
c031 8589 ff48 ffff c5e9 fffd baff 000a 0000 02e9 fffe baff 0009 0000 f8e9 fffd baff 0008 0000 eee9 fffd baff 0007 0000 e4e9 fffd 83ff 74f8 850f fdd6 ffff bd8b ff24 ffff ff85
3a75 758b 8d20 7895 ffff b9ff 98a2 0804 1489 bb24 0001 0000 9d89 ff24 ffff 7489 0824 4c89 0424 1de8 fffc 48ff 06ba 0000 0f00 5584 fffd e9ff fd97 ffff 05ba 0000 e900 fd8d ffff
b68d 0000 0000 bd8b      ffff ff85 8f0f 0149 0000 9d8d ff64 ffff abb8 0498 8d08      ffff           8d10 7895 ffff beff 0001 0000 b589 ff28 ffff 9d8d ff68      4c89 0c24 5c89
0824 4489 0424 1489      fbae ffff f883 0f03 f685 0000 fc00 bbbf 0498 b908 0003      de89           00d7 0000 858b ff60 ffff c085 8e0f 00bf 0000 bd8b ff64      ff85 8e0f 00b1
0000 28b9 0000 8300                8900                     4be8                                         0000      0424 8947                                              89ff
2474 8904 241c 15e8                ffff                                                                                 8bff                                              851e
74c0 8b1a 4085 ffff                85ff                                                                                 baff                                              31ff
e9d2 fc29 ffff 0bba                ffff                                                                                 baff                                              12ba
0000 3900 1c9d ffff                8bff                                                                                 3485                                              ff48
ffff                               38b5                                                                                 b704                                              3c8d
ffff                               ffff                                                                                 ff58                                              9d8b
ff34 ffff 308b 858b ff3c ffff de29 9d8b ff44 ffff fec1 8903 18b5 ffff 8bff 833c f739 8d0f 009a 0000 9d8b ff40 ffff 0c8b 3bbb 3c8d ffff 89ff 0fc8 7684 0000 8b00 349d ffff 8dff
fb34 b589 ff30 ffff ce89 958b ff44 ffff 9d8b ff34 ffff 0c8b 8b82 3085 ffff 8dff cb14 408b 8b04 045a 4289 8b04 3085 ffff 89ff 0458 008b 1a8b 0289 858b ff30 ffff 958b ff40 ffff
1889 048b 898a ba04 858b ff44 ffff 3489 ff8a b004 348b 3bba 3cb5 ffff 89ff 75f0 8ba3 3c8d ffff 89ff 47da bd3b ff18 ffff 8c0f ff66 ffff 85ff ff3c ffff bd8b ff48 ffff 8583 ff14
ffff 3928 3cbd ffff 0fff 178c ffff 89ff 5895 ffff 8bff 649d ffff 89ff 508d ffff 8bff 0c7d b58b ff48 ffff 4d8b 8908 8b1f 4c9d ffff 29ff 46de 5d8b 8918 8b31 5c8d ffff 8bff 38b5
ffff 8dff 8914 3c8d 8bd6 1c55 3b89 9d8b ff4c ffff 048d 899b 891a 8ddf c61c 758b 3910 8bf9 3485 ffff 89ff 8b1e 145d 0389 087c 8d3b ff48 ffff 0a7e 13ba 0000 e900 f9bb ffff 558b
8b18 8b02 2848 0839 ea74 8d8b ff44 ffff 0c89 e824 f826 ffff bd8b ff40 ffff 3c89 e824 f818 ffff c481 011c 0000 c031 5e5b 5d5f ebc3 900d 9090 9090 9090 9090 9090 9090 a155 aea8
0804 e589 8557 8bc0 a03d 04ae 5608 0f53 b28e 0001 c700 2047 0000 0000 4f8b 4818 a8a3 04ae 8b08 1057 f939 0d89 aed8 0804 1589 aeb4 0804 840f 01b6 0000 5f8b 8514 89d2 bc1d 04ae
0f08 8b85 0001 8900 1459 4b89 a118 aed8 0804 c085 840f 016e 0000 d0a3 04ae 8b08 1850 358b aeac 0804 1589 aec0 0804 0d8b aed0 0804 518b 8914 cc0d 04ae 8908 b815 04ae 8d08 0076
518b 891c 8bd3 9614 1589 aedc 0804 d285 840f 0098 0000 428b 3904 0441 8d0f 0084 0000 1589 aec8 0804 0d89 aedc 0804 0d8b aec8 0804 518b c714 2041 0001 0000 418b 8918 bc15 04ae
a308 aec4 0804 4289 8918 1450 dca1 04ae 8b08 1050 40ff 891c 0c41 d285 1589 aeb4 0804 2675 4989 8918 1449 4889 a110 aedc 0804 d0a3 04ae c708 9e04 0000 0000 0d8b aed0 0804 71e9
ffff 90ff 428b a314 aebc 0804 4a89 8914 1851 4189 8914 1848 cdeb 0d89 aec8 0804 80eb 0c89 8b9e c00d 04ae 3908 cc0d 04ae 8908 d41d 04ae 7408 8b11 b81d 04ae 8908 d01d 04ae e908
ff12 ffff 1d8b aeb0 0804 d231 ffb9 ffff 893f a40d 04ae 8908 d415 04ae 8508 7edb b948 0001 0000 448b fc8e c085 2d74 dca3 04ae 3108 89d2 8e54 a1fc aedc 0804 508b 3b04 a415 04ae
7d08 890b a415 04ae a308 aea0 0804 40c7 000c 0000 8900 39c8 8dd8 0149 c27c d4a3 04ae 5b08 f889 5f5e c35d f631 3589 aea0 0804 efeb 728b 8918 c435 04ae 8908 1451 4a89 8918 145e
```

and so on...

...in machine code

Implementing Dijkstra's algorithm

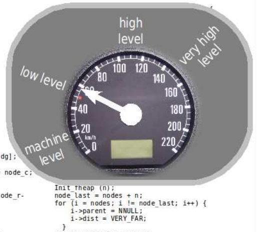


```
.L3:
    addl  $1, -148(%ebp)
    movzbl        -116(%ebp),
%eax
    movsbl        %al,%eax
    movl  %eax, -268(%ebp)
    cmpl  $100, -268(%ebp)
    je    .L7
    cmpl  $100, -268(%ebp)
    jg    .L11
    cmpl  $97, -268(%ebp)
    je    .L6
    cmpl  $97, -268(%ebp)
    jg    .L12
    cmpl  $0, -268(%ebp)
    je    .L2
    cmpl  $10, -268(%ebp)
    je    .L2
    jmp   .L4
.L12:
    cmpl  $99, -268(%ebp)
    je    .L2
    jmp   .L4
.L11:
    cmpl  $112, -268(%ebp)
    je    .L9
    cmpl  $116, -268(%ebp)
    je    .L10
    cmpl  $110, -268(%ebp)
    je    .L8
    jmp   .L4
.L9:
    cmpl  $0, -144(%ebp)
    jle   .L13
    movl  $0, -124(%ebp)
    jmp   .L15
.L13:
    movl  $1, -144(%ebp)
    leal  -220(%ebp), %eax
    movl  %eax, 16(%esp)
    leal  -204(%ebp), %eax
    movl  %eax, 12(%esp)
    leal  -119(%ebp), %eax
    movl  %eax, 8(%esp)
    movl  $.LC21, 4(%esp)
    leal  -116(%ebp), %eax
    movl  %eax, (%esp)
    call  sscanf
    cmpl  $3, %eax
    je    .L16
    movl  $1, -124(%ebp)
    jmp   .L15
.L16:
    leal  -119(%ebp), %eax
    movl  %eax, -276(%ebp)
    movl  $.LC22, -280(%ebp)
    movl  $3, -284(%ebp)
    cld
    movl  -276(%ebp), %esi
    movl  -280(%ebp), %edi
    movl  -284(%ebp), %ecx
    repz
    cmpsb
    seta  %dl
    setb  %al
    movl  %edx, %ecx
    subb  %al, %cl
    movl  %ecx, %eax
    movsbl        %al,%eax
    testl %eax, %eax
    je    .L18
    movl  $2, -124(%ebp)
    jmp   .L15
.L18:
    movl  -204(%ebp), %eax
    testl %eax, %eax
    jle   .L20
    movl  -220(%ebp), %eax
    testl %eax, %eax
    jg    .L22
.L20:
    movl  $3, -124(%ebp)
    jmp   .L15
.L22:
    movl  -204(%ebp), %eax
    addl  $2, %ea
    movl  $40, 4(
    movl  %eax, (
    call  calloc
    movl  %eax, -
    movl  -220(%e
    addl  $1, %ea

    call  calloc
    movl  %eax, -160(%ebp)
    movl  -220(%ebp), %eax
    movl  $4, 4(%esp)
    movl  %eax, (%esp)
    call  calloc
    movl  %eax, -188(%ebp)
    movl  -204(%ebp), %eax
    addl  $2, %eax
    movl  $4, 4(%esp)
    movl  %eax, (%esp)
    call  calloc
    movl  %eax, -192(%ebp)
    movl  -220(%ebp), %edx
    movl  -204(%ebp), %eax
    leal  (%edx,%eax), %eax
    leal  2(%eax), %edx
    movl  -220(%ebp), %eax
    leal  1(%eax), %ecx
    movl  -204(%ebp), %eax
    addl  $2, %eax
    movl  $4, 24(%esp)
    movl  %edx, 20(%esp)
    movl  $8, 16(%esp)
    movl  %ecx, 12(%esp)
    movl  $40, 8(%esp)
    movl  %eax, 4(%esp)
    movl  $.LC23, (%esp)
    call  printf
    cmpl  $0, -168(%ebp)
    je    .L23
    cmpl  $0, -160(%ebp)
    je    .L23
    cmpl  $0, -192(%ebp)
    je    .L23
    cmpl  $0, -188(%ebp)
    jne   .L27
.L23:
    movl  $4, -124(%ebp)
    jmp   .L15
.L27:
    movl  -160(%ebp), %eax
    movl  %eax, -156(%ebp)
    jmp   .L2
.L10:
    cmpl  $0, -140(%ebp)
    je    .L28
    movl  $5, -124(%ebp)
    jmp
L28:
    movl  -14
    movl  64(%e
    movl  ax,
    movl
    leal  -1
    movl  %eax, (%esp)
    call  sscanf
    cmpl  $1, %eax
    je    .L2
    movl  $6, -124(%ebp)
    jmp   .L15
.L8:
    cmpl  $0, -144(%ebp)
    jne   .L32
    movl  $7, -124(%ebp)
    jmp   .L15
.L32:
    cmpl  $0, -136(%ebp)
    je    .L34
    movl  $8, -124(%ebp)
    jmp   .L15
.L34:
    movl  $1, -136(%ebp)
    leal  -208(%ebp), %eax
    movl  %eax, 8(%esp)
    movl  $.LC25, 4(%esp)
    leal  -116(%ebp), %eax
    movl  %eax, (%esp)
    call  sscanf
    movl  %eax, -128(%ebp)
    cmpl  $0, -128(%ebp)
    jg    .L36
    movl  $9, -124(%ebp)
    jmp   .L15
.L36:
    movl  -208(%ebp), %eax
    testl %eax, %eax
    js    .L38
    movl  -208(%ebp), %edx
    movl  -204(%ebp), %eax
    cmpl  %eax, %edx
    jle   .L40
.L38:
    movl  $10, -124(%ebp)
    jmp   .L15
.L40:
    movl  $0, -196(%ebp)
    movl  -204(%ebp), %eax
    movl  %eax, -200(%ebp)
    jmp   .L2
.L7:
    movl  $11, -124(%ebp)
    jmp   .L15

    cmpl  %eax, -132(%ebp)
    jl    .L43
    movl  $13, -124(%ebp)
    jmp   .L15
.L43:
    leal  -224(%ebp), %eax
    movl  %eax, 16(%esp)
    leal  -212(%ebp), %eax
    movl  %eax, 12(%esp)
    leal  -216(%ebp), %eax
    movl  %eax, 8(%esp)
    movl  $.LC26, 4(%esp)
    leal  -116(%ebp), %eax
    movl  %eax, (%esp)
    call  sscanf
    cmpl  $3, %
    je    .L4
    movl
    jmp
.L45:

    jle
.L47:
    movl
    jmp
.L51:
    movl  -192(%ebp), %edx
    addl  $4, %edx
    movl  -216(%ebp), %eax
    sall  $2, %eax
    addl  %eax, %edx
    movl  (%edx), %eax
    addl  $1, %eax
    movl  %eax, (%edx)

    sall  $2, %eax
    addl  %edx, %eax
    sall  $3, %eax
    movl  %eax, %edx
    addl  -168(%ebp), %edx
    movl  -156(%ebp), %eax
    movl  %edx, 4(%eax)
    movl  -224(%ebp), %edx
    movl  -156(%ebp), %eax
    movl  %edx, (%eax)
    movl  -212(%ebp), %eax
    cmpl  -200(%ebp), %eax
    jge   .L52
    movl  -212(%ebp), %eax
    movl  %eax, -200(%ebp)

    movl  $16, -124(%ebp)
    jmp   .L15
.L2:
.L88:
    leal  -116(%ebp), %eax
    movl  %eax, (%esp)
    call  gets
    testl %eax, %eax
    jne   .L3
    movl  stdin, %eax
      l   %eax, (%esp)
      l   feof
     stl  %eax, %eax
     ne   .L61
     ovl  $17, -124(%ebp)
    jmp   .L15
 61:
    cmpl  $0, -148(%ebp)
    jne   .L63
    movl  $20, -124(%ebp)
    jmp   .L15
.L63:
    movl  -220(%ebp), %eax
    cmpl  %eax, -132(%ebp)
    jge   .L65
    movl  $18, -124(%ebp)
    jmp   .L15
```

and so on...

...in assembly

Implementing Dijkstra's algorithm



```c
#define nod(node) (long)(node-nodes+1)
#define VERY_FAR  1073741823
#define NNULL     (node*)NULL
typedef struct fheap_st {
  node *min;
  long dist;
  long n;
  node **deg_pointer;
  long deg_max;
} f_heap;
f_heap fh;
node *after, *before, *father, *child,
*first, *last,
  *node_c, *node_s, *node_r, *node_n,
*node_l;
long dg;
#define BASE           1.61803
#define NOT_ENOUGH_MEM 2
#define OUT_OF_HEAP    0
#define IN_HEAP        1
#define MARKED         2
#define NODE_IN_FHEAP( node ) ( node ->
status > OUT_OF_HEAP )
void Init_fheap (n) long n; {
  fh.deg_max = (long) (log ((double) n) /
log (BASE) + 1);
  if ((fh.deg_pointer = (node **) calloc
(fh.deg_max, sizeof (node *)))
      == (node **) NULL)
    exit (NOT_ENOUGH_MEM);
  for (dg = 0; dg < fh.deg_max; dg++)
    fh.deg_pointer[dg] = NNULL;
  fh.n = 0;
  fh.min = NNULL;
}
void Check_min (nd) node *nd; {
  if (nd->dist < fh.dist) {
      fh.dist = nd->dist;
      fh.min = nd;
    }
}
void Insert_after_min (nd) node *nd; {
  after = fh.min->next;
  nd->next = after;
  after->prev = nd;
  fh.min->next = nd;
  nd->prev = fh.min;
  Check_min (nd);
}
void Insert_to_root (nd) node *nd; {
  nd->heap_parent = NNULL;
  nd->status = IN_HEAP;
  Insert_after_min (nd);
}
void Cut_node (nd, father) node *nd,
*father; {
  after = nd->next;
  if (after != nd) {
      before = nd->prev;
      before->next = after;
      after->prev = before;
    }
  if (father->son == nd) father->son =
after;
  (father->deg)--;
  if (father->deg == 0) father->son =
NNULL;
}
void Insert_to_fheap (nd) node *nd; {
  nd->heap_parent = NNULL;
  nd->son = NNULL;
  nd->status = IN_HEAP;
  nd->deg = 0;
  if (fh.min == NNULL) {
      nd->prev = nd->next = nd;
      fh.min = nd;
      fh.dist = nd->dist;
    } else Insert_after_min (nd);
  fh.n++;
}
void Fheap_decrease_key (nd) node *nd; {
  if ((father = nd->heap_parent) == NNULL)
    Check_min (nd);
  else {
      if (nd->dist < father->dist) {
     node_c = nd;
     while (father != NNULL) {
        Cut_node (node_c, fath
        Insert_to_root (node_
        if (father->status ==
       father->status = MARKED
       break;
      }
        node_c
        father
      }
    }
  }
}
node * Extract_min () {
  node *nd;
  nd = fh.min;
  if (fh.n > 0) {
      fh.n--;
      fh.min->status = OUT_OF_HEAP;
      first = fh.min->prev;
      child = fh.min->son;
      if (first == fh.min) first = child;
      else {
     after = fh.min->next;
      if (child == NNULL) {
        first->next = after;
        after->prev = first;
      } else {
        before = child->prev;
        first->next = child;
        child->prev = first;
        before->next = after;
        after->prev = before;
      }
    }
     if (first != NNULL) {
      node_c = first;
      last = first->prev;
      while (1) {
        node_l = node_c;
        node_n = node_c->next;
        while (1) {
       dg = node_c->deg;
       node_r = fh.deg_pointer[dg];
       if (node_r == NNULL) {
          fh.deg_pointer[dg] = node_c;
          break;
         } else {
          if (node_c->dist < node_r-
>dist) {
         node_s = node_r;
         node_r = node_c;
        } else node_s = node_c;
          after = node_s->next;
          before = node_s->prev;
                prev = before;
                ext = after;
          node_r-   g++;
          node_s->heap_parent = node_r;
          node_s->status = IN_HEAP;
          child = node_r->son;
          if (c      == NNULL)
         no          node_s->next =
            node_s->prev       s;
          else {
         after = child->next;
         child->next = node_s;
         node_s->prev = child;
         node_s->next = after;
         after->prev = node_s;
        }
       }
       node_c = node_r;
       fh.deg_pointer[dg] = NNULL;
      }
        if (node_l == last) break;
        node_c = node_n;
      }
    fh.dist = VERY_FAR;

  Init_fheap (n);
  node_last = nodes + n;
  for (i = nodes; i != node_last; i++) {
      i->parent = NNULL;
      i->dist = VERY_FAR;
    }
  source->parent = source;
  source->dist = 0;
  Insert_to_fheap (source);
  while (1) {
      node_from = Extract_min ();
      if (node_from == NNULL) break;
      num_scans++;
      arc_last = (node_from + 1)->first;
      dist_from = node_from->dist;
      for (arc_ij = node_from->first;
arc_ij != arc_last; arc_ij++) {
    node_to = arc_ij->head;
    dist_new = dist_from + (arc_ij->len);
    if (dist_new < node_to->dist) {
        node_to->dist = dist_new;
        node_to->parent = node_from;
        if (NODE_IN_FHEAP (node_to))
Fheap_decrease_key (node_to);
        else Insert_to_fheap (node_to);
      }
    }
  }
  n_scans = num_scans;
  return (0);
}
```

...in C

Implementing Dijkstra's algorithm

```java
public final class DijkstraShortestPath<V, E> {
  private List<E> edgeList;
   private double pathLength;
   public DijkstraShortestPath(Graph<V, E> graph,
       V startVertex, V endVertex) {
          this(graph, startVertex, endVertex,
           Double.POSITIVE_INFINITY);
     }
   public DijkstraShortestPath(Graph<V, E> graph,
     V startVertex, V endVertex, double radius) {
      CFIterator<V, E> iter = new
      CFIterator<V,E>(graph, startVertex, radius);
     while (iter.hasNext()) {
         V vertex = iter.next();
         if (vertex.equals(endVertex)) {
          createEdgeList(graph, iter, endVertex);
          pathLength =
            iter.getShortestPathLength(endVertex);
          return;
         }
     }
     edgeList = null;
     pathLength = Double.POSITIVE_INFINITY;
  }
 public List<E> getPathEdgeList() {
  return edgeList;
 }
}
public class FHNode<T> {
  T data;
  FHNode<T> child;
  FHNode<T> left;
  FHNode<T> parent;
  FHNode<T> right;
  boolean mark;
  double key;
  int degree;
  public FHNode(T data, double key) {
      right = this;
      left = this;
      this.data = data;
      this.key = key;
  }
  public final double getKey() {
      return key;
  }
  public final T getData() {
      return data;
  }
}
public class FH<T> {
  private static final double oneOverLogPhi
   = 1.0 / Math.log((1.0 + Math.sqrt(5.0)) / 2.0);
  private FHNode<T> minNode;
  private int nNodes;
  public FH() {}
  public boolean isEmpty() {
      return minNode == null;
  }
  public void clear() {
      minNode = null;
      nNodes = 0;
  }
  public void decreaseKey(FHNode<T> x, double k) {
      x.key = k;
      FHNode<T> y = x.parent;
      if ((y != null) && (x.key < y.key)) {
          cut(x, y);
          cascadingCut(y);
      }
      if (x.key < minNode.key) {
          minNode = x;
      }
  }
  public void delete(   ode<T> x) {
      decreaseKey(x,    ble NEGATIVE_INFINITY);
           eMin();
  }
  p   ic     d insert(

        node.right = minNode.right;
        minNode.right = node;
        node.right.left = node;
        if (key < minNode.key) minNode = node;
      } else minNode = node;
      nNodes++;
  }
  public FHNode<T> removeMin() {
      FHNode<T> z = minNode;
      if (z != null) {
          int numKids = z.degree;
          FHNode<T> x = z.child;

          }
          nNodes--;
      }
      return z;
  }
  protected void cascadingCut(FHNode<T> y) {
      FHNode<T> z = y.parent;
      if (z != null) {
          if (!y.mark) {
              y.mark = true;
          } else {
              cut(y, z);
              cascadingCut(z);
          }
      }
  }
```

and so on...

...in Java

Implementing Dijkstra's algorithm

```
dijkstra(Vertex, Ss):-
  create(Vertex, [Vertex], Ds),
  dijkstra_1(Ds, [s(Vertex,0,[])], Ss).
dijkstra_1([], Ss, Ss).
dijkstra_1([D|Ds], Ss0, Ss):-
  best(Ds, D, S),
  delete([D|Ds], [S], Ds1),
  S=s(Vertex,Distance,Path),
  reverse([Vertex|Path], Path1),
  merge(Ss0, [s(Vertex,Distance,Path1)], Ss1),
  create(Vertex, [Vertex|Path], Ds2),
  delete(Ds2, Ss1, Ds3),
  incr(Ds3, Distance, Ds4),
  merge(Ds1, Ds4, Ds5),
  dijkstra_1(Ds5, Ss1, Ss).

path(Vertex0, Vertex, Path, Dist):-
  dijkstra(Vertex0, Ss),
  member(s(Vertex,Dist,Path), Ss), !.

create(Start, Path, Edges):-
  setof(s(Vertex,Edge,Path),
  e(Start,Vertex,Edge), Edges), !.
create(_, _, []).

best([], Best, Best).
best([Edge|Edges], Best0, Be
  shorter(Edge, Best0), !,
  best(Edges, Edge, Best).
best([_|Edges], Best0, Best):-
  best(Edges, Best0, Best).

shorter(s(_,X,_), s(_,Y,_)):-X < Y.

delete([], _, []).
delete([X|Xs], [], [X|Xs]):-!.
delete([X|Xs], [Y|Ys], Ds):-
  eq(X, Y), !,
```

```
delete([X|Xs], [Y|Ys], [X|Ds]):-
  lt(X, Y), !, delete(Xs, [Y|Y
delete([X|Xs], [_|Ys], Ds):-
  delete([X|Xs], Ys, Ds).

merge([], Ys, Ys).
merge([X|Xs], [], [X|Xs]):
merge([X|Xs], [Y|Ys], [X|Z
  eq(X, Y), shorter(X, Y),
  merge(Xs, Ys, Zs).
merge([X|Xs], [Y|Ys], [Y|Z
  eq(X, Y), !,
  merge(Xs, Ys, Zs).
merge([X|Xs], [Y|Ys], [X|Zs],
  lt(X, Y), !,
  merge(Xs, [Y|Ys], Zs).
merge([X|Xs], [Y|Ys], [Y|Zs]):-
  merge([X|Xs], Ys, Zs).

eq(s(X,_,_), s(X,_,_)).
lt(s(X,_,_), s(Y,_,_)):-X @< Y.

  ([], _, []).
in                        P)|Ys]):-

  In   X   Inc

membe  X,
member(X, [_|Ys]):-member(X,

reverse(Xs, Ys):-reverse_1(Xs, [], Ys).
reverse_1([], As, As).
reverse_1([X|Xs], As, Ys):-reverse_1(Xs, [X|As], Ys).

e(X, Y, Z):-dist(X, Y, Z).
e(X, Y, Z):-dist(Y, X, Z).
```

...in Prolog

Implementing Dijkstra's algorithm

```
:- chr_constraint edge(+node,+node,+length), dijkstra(+node),
        distance(+node,+length), scan(+node,+length),
        relabel(+node,+length).
:- chr_type node == int.
:- chr_type length == number.

dijkstra(A) <=> scan(A,0).
scan(N,L), edge(N,N2,W) ==> L2 is L+W,relabel(N2,L2).
scan(N,L) <=> distance(N,L),
        (extract_min(N2,L2) -> scan(N2,L2) ; true).
distance(N,_) \ relabel(N,_) <=> true.
relabel(N,L) <=> decr_or_ins(N,L).


:- chr_constraint insert(+item,+key), extract_min(?item,?key),
              decr_or_ins(+item,+key), decr(+item,+key),
   mark(+item), ch2rt(+item), decr(+item,+key,+item,+item,+mark),
   findmin, min(+item,+key), item(+item,+key,+item,+item,+mark).

:- chr_type item == int.
:- chr_type key == number.
:- chr_type mark ---> m ; u.

insert(I,K) <=> item(I,K,0,0,u), min(I,K).

min(_,A) \ min(_,B) <=> A =< B | true.

extract_min(X,Y), min(I,K), item(I,_,_,_,_)
            <=> ch2rt(I), findmin, X=I, Y=K.
extract_min(_,_) <=> fail.

ch2rt(I) \ item(C,K,R,I,_)#passive
            <=> item(C,K,R,0,u).
ch2rt(I) <=> true.
```

```
findmin, item(I,K,_,
findmin <=> true.

item(I1,K1,R,0,_), i
          <=> K1 < K

           ; item(I1

decr(I,K), item(I,0,R
           <=> K < 0 |
decr(I,K) <=> fail.

item(I,0,R,P,M), decr_or_ins(I,K)
           <=> K < 0 | decr(I,K,R,P,M).
item(I,0,_,_,_) \ decr_or_ins(I,K) <=> K >= 0 | true.
decr_or_ins(I,K) <=> insert(I,K).

   r(I,                min(I,K).
    I,  R,             item(I,K,R,0,u).
item(P,             \  ecr(I,K,R,P,M)
                      PK | item(I,K,R,P,M).
dec  I,  R,  M         item(I,K,R,0,u), mark(P).

mark(I), item(I,K,R,0,_) <=> item(I,K,R-1,0,u).
mark(I), item(I,K,R,P,m)
           <=> item(I,K,R-1,0,u), mark(P).
mark(I), item(I,K,R,P,u) <=> item(I,K,R-1,P,m).
mark(I) <=> writeln(error_mark), fail.
```

...in CHR

Implementing Dijkstra's algorithm

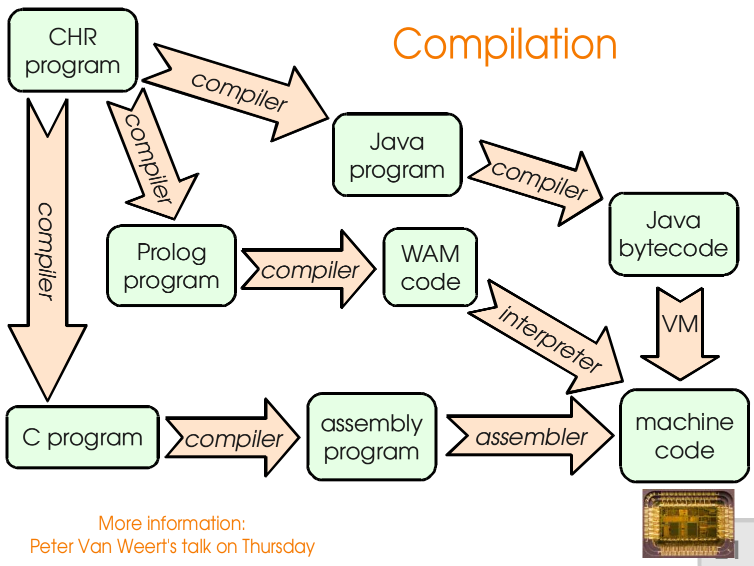
```
:- chr_constraint   edge(+node,+node,+length),
                    source(+node),
                    distance(+node,+length).
:- chr_type node == int.
:- chr_type length == number.

1 :: source(V) ==> distance(V,0).
1 :: distance(V,D1) \ distance(V,D2) <=> D1 =< D2 | true.
D+2 :: distance(V,D), edge(V,C,W) ==> distance(W,D+C).
```

...in CHR^{rp}

CHR = Constraint Handling Rules

- CHR is a very **high level** programming language
- based on **rules**
  - propagation rules:
    - clouds ⇒ forecast(rainy).
    - forecast(rainy) ⇒ bring(coat).
    - forecast(sunny) ⇒ bring(sunscreen).
  - simplification rules:
    - bring(coat), bring(sunscreen) ⇔ bring(umbrella).
- stand-alone (CHR-only) or extending a **host language**

驰



Compilation

CHR program

compiler

Java program

compiler

Java bytecode

VM

compiler

Prolog program

compiler

WAM code

interpreter

compiler

C program

compiler

assembly program

assembler

machine code

More information:
Peter Van Weert's talk on Thursday

Syntax of CHR

head: CHR constraints
guard: host language (built-in)
body: CHR constraints + host language

- Propagation rule:

  head ⇒ guard | body.

  example: dist(A,D), road(A,B,L) ⇒ dist(B,D+L).

- Simplification rule:

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

Operational semantics of CHR

IF head IN STORE (AND guard HOLDS), THEN...

- Propagation rule: **... ADD body TO STORE**

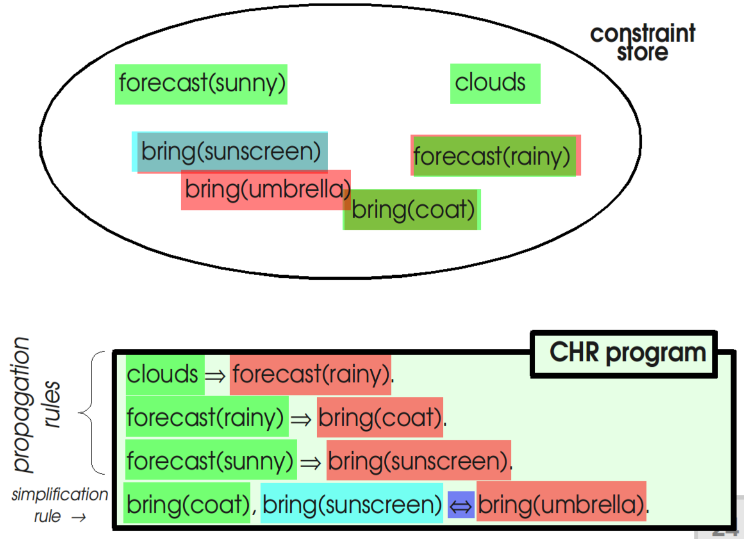
  head ⇒ guard | body.

  example: dist(A,D), road(A,B,L) ⇒ dist(B,D+L).

- Simplification rule: **... REPLACE head BY body**

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).



constraint store

forecast(sunny)

clouds

bring(sunscreen)

forecast(rainy)

bring(umbrella)

bring(coat)

CHR program

propagation rules

clouds ⇒ forecast(rainy).

forecast(rainy) ⇒ bring(coat).

forecast(sunny) ⇒ bring(sunscreen).

simplification rule →

bring(coat), bring(sunscreen) ⇔ bring(umbrella).

Features of CHR

Embedded in a host language

CHR *extends* an existing programming language, e.g.
CHR(Prolog)
CHR(Haskell)
CHR(Java)
CHR(C)

⇒ dist(B,D+L).

- Simplification rule:

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

Features of CHR

Multiple heads

The head of a rule consists of an arbitrary number of CHR constraints (1 or more)

cf. Prolog: single-headed

- Propagation rule:

  head ⇒ guard | body.

  example: dist(A,D), road(A,B,L) ⇒ dist(B,D+L).

- Simplification rule:

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

Features of CHR

- Propagation rule:

  head ⇒ guard |

  example: dist(A,D), ro

Multi-set semantics

The constraint store may contain the same constraint multiple times
{c} is not the same as {c,c}

cf. classical logic: p ↔ p ∧ p

- Simplification rule:

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

Features of CHR

Important remark:

in CHR(Prolog), we can still use Prolog disjunction or nondeterministic predicates in the body of rules!

CHR with disjunction/search is called **CHR**$^{\vee}$

- Propagation rule:

Committed-choice

Once a rule has been applied, it remains applied – no backtracking to try different derivation paths

cf. Prolog: choice-points and backtracking

d(A,B,L) ⇒ dist(B,D+L).

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

Features of CHR

Logical semantics

CHR has a declarative semantics!

- Propagation rule:

  propagation = implication

  head ⇒ guard | body.

  example: dist(A,D), road(A,B,L) ⇒ dist(B,D+L).

- Simplification rule:

  simplification = equivalence

  head ⇔ guard | body.

  example: dist(A,X), dist(A,Y) ⇔ X≤Y | dist(A,X).

PART TWO

Writing CHR programs

CHR(Prolog) by example

- Simple example: color mixing in CHR
- We first declare CHR constraints as follows:

```
:- chr_constraint red, blue, yellow, purple, …
```

- Then we write the rules:

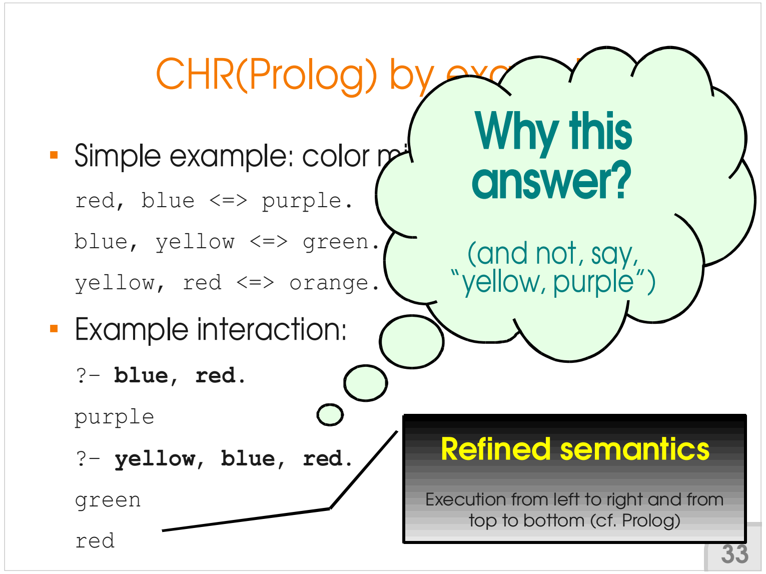
```
red, blue <=> purple.
blue, yellow <=> green.
yellow, red <=> orange.
```

CHR(Prolog) by example

- Simple example: color mixing in CHR

```
red, blue <=> purple.
blue, yellow <=> green.
yellow, red <=> orange.
```

- CHR program execution:
  - user gives a **goal**
  - rules are applied exhaustively
  - the remaining constraints are the **result**



CHR(Prolog) by exa

- Simple example: color m

```
red, blue <=> purple.
blue, yellow <=> green.
yellow, red <=> orange.
```

- Example interaction:

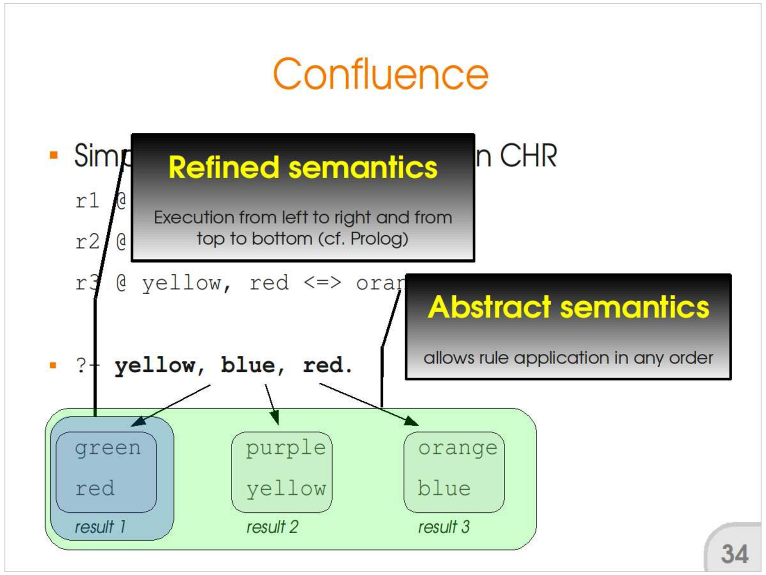
```
?- blue, red.
purple
?- yellow, blue, red.
green
red
```

Why this answer?

(and not, say, "yellow, purple")

Refined semantics

Execution from left to right and from top to bottom (cf. Prolog)



Confluence

- Simp n CHR

```
r1 @
r2 @
r3 @ yellow, red <=> oran
```

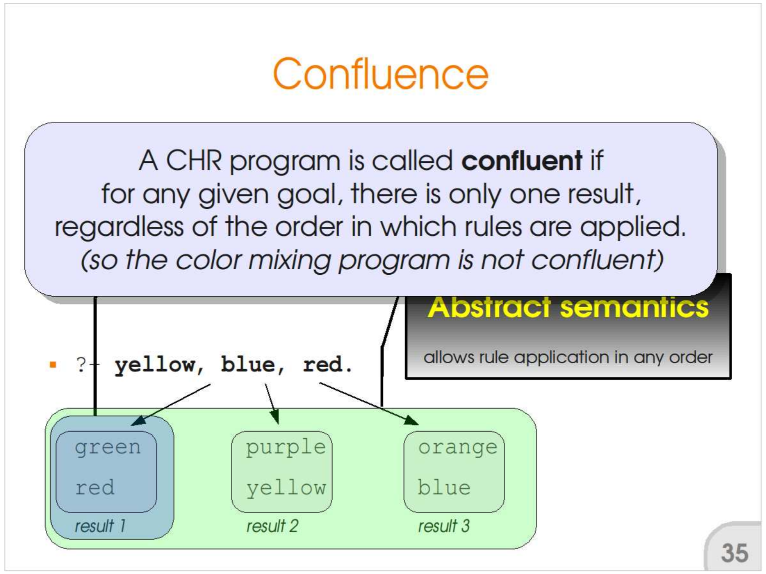
- `?- yellow, blue, red.`

Refined semantics

Execution from left to right and from top to bottom (cf. Prolog)

Abstract semantics

allows rule application in any order



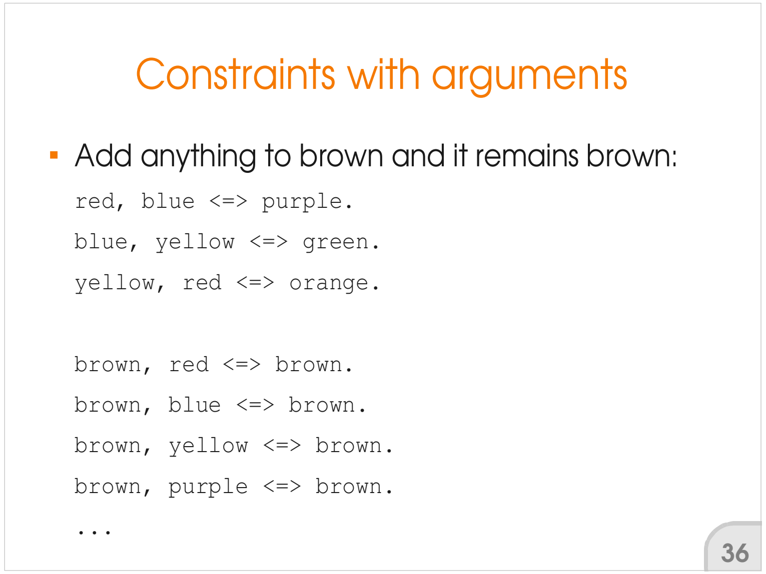
Confluence

A CHR program is called **confluent** if
for any given goal, there is only one result,
regardless of the order in which rules are applied.
(so the color mixing program is not confluent)

Abstract semantics

allows rule application in any order

- `?- yellow, blue, red.`



Constraints with arguments

- Add anything to brown and it remains brown:

```
red, blue <=> purple.
blue, yellow <=> green.
yellow, red <=> orange.

brown, red <=> brown.
brown, blue <=> brown.
brown, yellow <=> brown.
brown, purple <=> brown.
...
```

Constraints with arguments

- This becomes a bit tedious, can't we write something like this instead?

```
brown, _ <=> brown.
```

- The above will not work in CHR (but it does work in a related formalism called ACD term rewriting)
- But we can write our program in a different way...

Constraints with arguments

- From many 0-ary constraints to one unary constraint:

```
:- chr_constraint red, blue, yellow, purple, ...
red, blue <=> purple.
blue, yellow <=> green.
yellow, red <=> orange.

:- chr_constraint color/1.
color(red), color(blue) <=> color(purple).
...
```

Constraints with arguments

- Now we can write more general rules:

```
:- chr_constraint color/1.
color(X), color(Y) <=> mix(X,Y,Z) | color(Z).
color(brown), color(_) <=> color(brown).

% host language
mix(red,blue,purple).
mix(blue,yellow,green).
mix(yellow,red,orange).
```

Type and mode declarations

- Optionally, we can specify types and modes:

```
% no type/mode declaration:
:- chr_constraint color/1.

% only mode declaration:
:- chr_constraint color(+).    % ground argument

% type and mode declaration:
:- chr_constraint color(+colorname).
:- chr_type colorname ---> red ; blue ; yellow ;...
```

Simpagation rules

- So far we have only used simplification rules.
- Simpagation rules can be more concise/efficient:

```
% simplification rule:
color(brown), color(_) <=> color(brown).
```

“`true`” ?

In Prolog, “`true`” is a built-in that does not do anything. We use it to indicate an empty body.

```
% simpagation rule:
color(brown) \ color(_) <=> true.
```

Typical pattern #1: flattening lists

- We want to convert “`colors([red,green,blue])`” to “`color(red), color(green), color(blue)`”

```
:- chr_constraint color(+colorname).
:- chr_type colorname ---> red ; blue ; yellow ;...
:- chr_constraint colors(+list(colorname)).
:- chr_type list(T) ---> [] ; [T|list(T)].

colors([]) <=> true.
colors([C|Rest]) <=> color(C), colors(Rest).
```

(just like how you would do this in Prolog)

More complex color mixing

- Now we also specify the amount of paint:

```
:- chr_constraint color(+colorname,+amount).
:- chr_type colorname ---> red ; blue ; yellow ;...
:- chr_type amount == float.     % in liters
```

(“`float`” is a built-in type for floating point numbers)

Typical pattern #2: “default constructor”

- For backwards compatibility, we still have `color/1`

```
:- chr_constraint color(+colorname).
:- chr_constraint color(+colorname,+amount).
:- chr_type colorname ---> red ; blue ; yellow ;...
:- chr_type amount == float.

% we assume 1 liter of paint:
color(C) <=> color(C,1).
```

Typical pattern #3: maintaining a sum

```
:- chr_constraint color(+colorname,+amount).

color(C,A1), color(C,A2)
      <=> TA is A1+A2, color(C,TA).

color(C,0) <=> true.
color(X,A1), color(Y,A2)
      <=> mix(X,Y,Z) | TA is A1+A2, color(Z,TA).
```

CHR(Prolog) one-liners (1)

- Finding the minimum:

```
min(A) \ min(B) <=> A =< B | true.

?- min(8), min(3), min(6), min(7).
min(3)
```

- Computing the sum:

```
sum(A), sum(B) <=> C is A+B, sum(C).

?- sum(3), sum(5), sum(6).
sum(14)
```

CHR(Prolog

- Finding the min

```
min(A) \ min(B)

?- min(8), min(3), mi
min(3)
```

- Computing the

```
sum(A), sum(B) <=>

?- sum(3), sum(5), su
sum(14)
```

Online algorithm

An online algorithm processes its inputs while they arrive (it does not need to see the full input to get started)

Using CHR often results in online algorithms

Anytime algorithm

An anytime algorithm can be interrupted during the computation to give a partial (approximate) result, from which it can then resume the computation

Using CHR often results in anytime algorithms

Concurrent algorithm

A concurrent algorithm can be executed in parallel (while sequential algorithms are hard to parallelize)

Using CHR often results in concurrent algorithms

CHR(Prolog) one-liners (2)

- Transitive closure

```
:- op(700,xfx,before).
:- chr_constraint before(+any,+any).

A before B, B before C ==> A before C.


?- a before b, b before c, c before d.
a before b
b before c
c before d
a before c
a before d
b before d
```

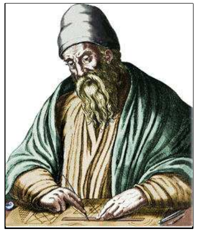
CHR(Prolog) one-liners (3)

- Naive merge-sort in *O(n2)* time

```
:- op(700,xfx,before).
:- chr_constraint before(+any,+any).

A before B \ A before C <=> B @< C | B before C.
```

```
?- 0 before foo, 0 before bar, 0 before baz, 0 before quux.
0 before bar
bar before baz
baz before foo
foo before quux
```

CHR(Prolog) two-liners (1)



- Greatest common divisor
  (Euclid's algorithm)

```
:- chr_constraint gcd(+int).

gcd(0) <=> true.

gcd(N) \ gcd(M) <=> N =< M | L is M mod N, gcd(L).
```

```
?- gcd(94017), gcd(1155), gcd(2035).
gcd(11)
```

CHR(Prolog) two-liners (2)

- Prime number generator
  (sieve of Eratosthenes)

```
:- chr_constraint prime(+int).

prime(N) ==> N>2 | M is N-1, prime(M).

prime(A) \ prime(B) <=> B mod A =:= 0 | true.
```

```
?- prime(10).
prime(2)
prime(3)
prime(5)
prime(7)
```

CHR(Prolog) two-liners (3)

- Fibonacci numbers

```
:- chr_constraint fib(+int,+int), upto(+int).

upto(_) ==> fib(0,1), fib(1,1).

upto(Max), fib(N1,M1), fib(N2,M2)
  ==> Max>N2, N2 is N1+1 |
    N is N2+1, M is M1+M2, fib(N,M).
```

```
?- upto(10).
fib(10,89)
fib(9,55)
fib(8,34)
fib(7,21)
fib(6,13)
...
```

CHR(Prolog) two-liners (4)

- Optimal merge-sort

```
:- op(700,xfx,before).
:- chr_constraint before(+any,+any), sort(+int,+any).

X before A  \  X before B
      <=> A @< B | A before B.


sort(N,A), sort(N,B)
      <=> A @< B | M is N+1, sort(M,A), A before B.


?- sort(0,foo), sort(0,bar), sort(0,baz), sort(0,quux).
bar before baz
baz before foo
foo before quux
sort(2,bar)
```

CHR(Prolog) two-liners (5)

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Soduko puzzle solver in CHR

```
:- chr_constraint given(+pos,+val), maybe(+pos,+list(val)).

given(P1,V) \ maybe(P2,L)
  <=> sees(P1,P2), select(V,L,L2) | maybe(P2,L2).

maybe(P,L) <=> member(V,L), given(P,V).

sees(X-_, X-_).                                       % same row
sees(_-X, _-X).                                       % same column
sees(X-Y, A-B) :- X//3 =:= A//3,  Y//3 =:= B//3.    % same box


?- given(1-1,5),given(1-2,3), ..., maybe(1-3,[1,2,3,...,9]), ...
given(a-1,5)
given(a-2,3)
given(a-3,7)
...
```

One last example... ≤

- Simple less-than-or-equal constraint solver

```
:- op(700,xfx,leq).
:- chr_constraint leq/2.
reflexivity @ X leq X <=> true.
idempotence @ X leq Y \ X leq Y <=> true.
antisymmetry @ X leq Y, Y leq X <=> X=Y.
transitivity @ X leq Y, Y leq Z ==> X leq Z.


?- A leq B, B leq C, C leq A.
A = B
B = C
```

Differences between CHR and Prolog

	Prolog	**CHR**
basic elements	predicates	constraints
elements are defined by	clauses	rules
syntax	`head :- body.`	`head <=> guard | body.`
#heads	1	1, 2, 3, ...
definition selection condition	unification	matching + guard
different applicable definitions	try alternatives (backtracking)	committed-choice
no applicable definition	failure	suspension (delay) ↳ partial result

Committed-choice – different from Prolog!

- In Prolog, **backtracking** (proof search) is used to find a non-failing derivation
- In CHR there is no backtracking
-
```
:- chr_constraint chr/0, output/1.
chr <=> output(foo).
chr <=> output(bar).
prolog :- output(foo).
prolog :- output(bar).
```

```
?- prolog.                          ?- chr.
output(foo) ;                       output(foo)

output(bar)
```

Head matching – different from Prolog!

- In Prolog, **unification** is used to match clause heads
- In CHR, **matching** (one-way unification) is used
-
```
:- chr_constraint chr/1, output/1.
chr(foo) <=> output(bar).
prolog(foo) :- output(bar).
```

```
?- prolog(foo).                     ?- chr(foo).
output(bar)                         output(bar)

?- prolog(Variable).                ?- chr(Variable).
output(bar)                         chr(Variable)
Variable = foo

?- prolog(quux).                    ?- chr(quux).
No                                  chr(quux)
```

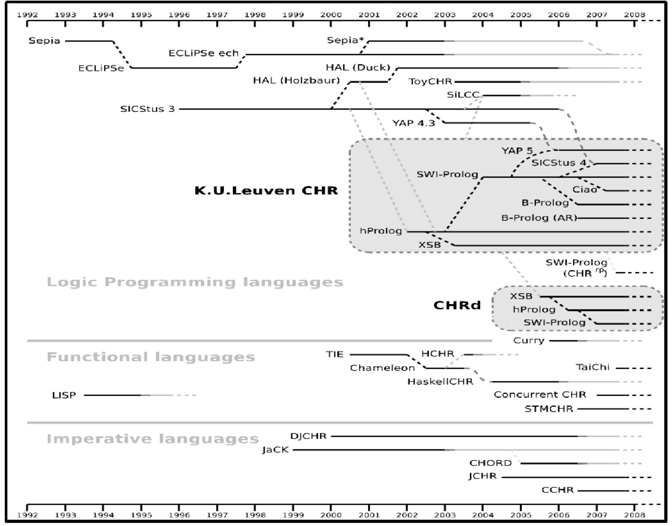
PART THREE

Theory & Applications

History of CHR: some milestones

1991 CHR is born, Thom Frühwirth
1995 Christian Holzbaur implements CHR(SICStus)
1998 confluence, program analysis (PhD Slim Abdennadher)
2002 Tom Schrijvers implements Leuven CHR system
2002- optimized compilation (PhDs Gregory Duck, Tom Schrijvers)
2003 First CHR book [Frühwirth&Abdennadher, Essentials of Constraint Programming]
2004 refined semantics, Gregory Duck et al.
2004 First CHR workshop
2005- computational complexity (PhD Jon Sneyers)
2005- Peter Van Weert implements Leuven JCHR (Java)
2007 Sulzmann & Lam implement first concurrent system
2009 Second CHR book, sixth CHR workshop

Many CHR systems...



Theory topics (1)

- Semantics
  - Declarative (logical) semantics
    - Classical logic (Frühwirth)
    - Linear logic (Hariolf Betz)
    - Transaction logic, …
    - Compositional semantics (Gabbrielli et al)
  - Operational semantics
    - Abstract semantics
    - Refined semantics (Duck et al)
    - Priority semantics (Leslie De Koninck)

Theory topics (2)

- Extensions / variants of CHR
  - Adaptive CHR (Armin Wolf)
  - Disjunction, search (Abdennadher, Wolf, De Koninck, …)
  - Negation, aggregates (Van Weert & Sneyers, …)
  - Modularity, solver hierarchies (Duck et al, Schrijvers et al, Fages et al)
  - Probabilistic CHR (Frühwirth et al, Sneyers et al)
  - …

Theory topics (3)

- Relationship to other formalisms
  - Term rewriting (ACD term rewriting, Duck, Stuckey et al)
  - Production rules / business rules (Van Weert)
  - Join-Calculus (Sulzmann and Lam)
  - Logical Algorithms (De Koninck)
  - Graph Transformation Systems (Raiser)
  - Petri nets (Betz)
  - …

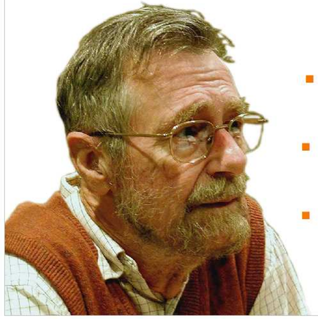
Theory topics (4)

- Program analysis
  - Confluence (Abdennadher, Duck et al, Raiser&Tacchella, Haemmerlé&Fages, ...)
  - Operational equivalence (Abdennadher&Frühwirth)
  - Termination (Frühwirth, Paolo Pilozzi, Dean Voets)
  - Complexity (Frühwirth&Schrijvers, Sneyers, De Koninck)
  - Abstract interpretation (Schrijvers, Stuckey, Duck)
  - ...

Back to the shortest path problem...

- How long does it take?
  - **It depends...**

how to find the shortest path ???



- which algorithm is used ?
- how is it implemented ?
- how large is the map (graph) ?

Computational Complexity Theory

- How does an algorithm **scale** with the input size?

	input size	*algorithm A* log-linear $O(n \log n)$	*algorithm B* quadratic $O(n^2)$
Leuven	5000	2 ms	25 ms
Brussels	50000	23 ms	2.5 seconds
New York City	277863	151 ms	1 min 17 seconds
Florida	1228116	747 ms	25 min, 8 seconds
North America	29883886	22 seconds	10 days, 8 hours, 4 min

What about Dijkstra's algorithm?

- Dijkstra's algorithm is $O(n \log n)$
  - for sparse graphs (in general: $O(m + n \log n)$)
  - if implemented in a good way, e.g. using Fibonacci-heaps
- This is optimal: you cannot do better
- Dijkstra's algorithm can be implemented in CHR
  (with the optimal complexity)

Some other examples...

Edsger Dijkstra (1930-2002)
Dutch computer scientist

Dijkstra's algorithm
can be implemented efficiently in CHR

Robert E. Tarjan (1948-)
American computer scientist

Jan van Leeuwen (1946-)
Dutch computer scientist

The Union-Find algorithm
can be implemented efficiently in CHR

John E. Hopcroft (1939-)
American computer scientist

Hopcroft's algorithm
can be implemented efficiently in CHR

?

... can **everything** be implemented efficiently in CHR?

Can we implement **everything** efficiently in CHR?

Complexity-wise completeness result for CHR

More information:
my talk on Thursday

Application domains

- Constraint solvers
  - CHR was specifically designed for this
  - Some domains where CHR has been used:
    - Scheduling
    - Soft constraints
    - Spatio-temporal reasoning
    - Multi-agent systems
    - Semantic web
- General-purpose programming language
  - Many classical algorithms have been implemented in CHR in a very elegant and natural way - often more concise than pseudocode!

Application domains

- Programming language development
  - Type systems (e.g. Haskell type classes)
  - Abductive reasoning
  - Computational linguistics (NLP)
    - CHR Grammars (Dahl&Christiansen)
  - Meta-programming
  - Testing & verification
- CHR can be used as a high-performance business rule engine (integrated in your favorite host language!)

Further reading...

- **Book:** Thom Frühwirth, *Constraint Handling Rules*, Cambridge University Press, July 2009.
- **Introductory survey:** Thom Frühwirth, *Theory and Practice of Constraint Handling Rules*, Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriott, Eds.), Journal of Logic Programming, Vol 37(1-3), October 1998.
- **Advanced survey:** Jon Sneyers, Peter Van Weert, Tom Schrijvers and Leslie De Koninck, *As Time Goes By: Constraint Handling Rules — A Survey of CHR Research from 1998 to 2007*, Theory and Practice of Logic Programming, 2009, To appear.