Synchronous programming

Critical Real Time Embedded Software

David Lesens

Wednesday, 06 October 2010

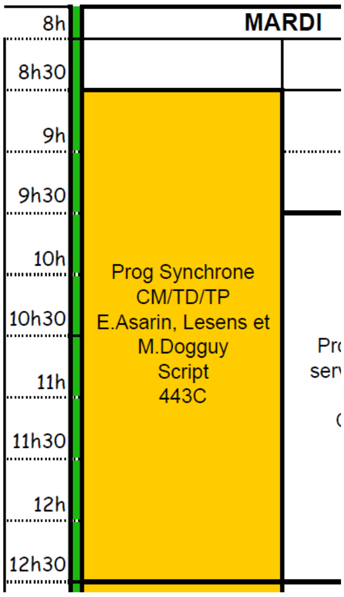
Synchronous programming

- Eugene Asarin
- Mehdi Dogguy

université PARIS DIDEROT PARIS 7

- David Lesens

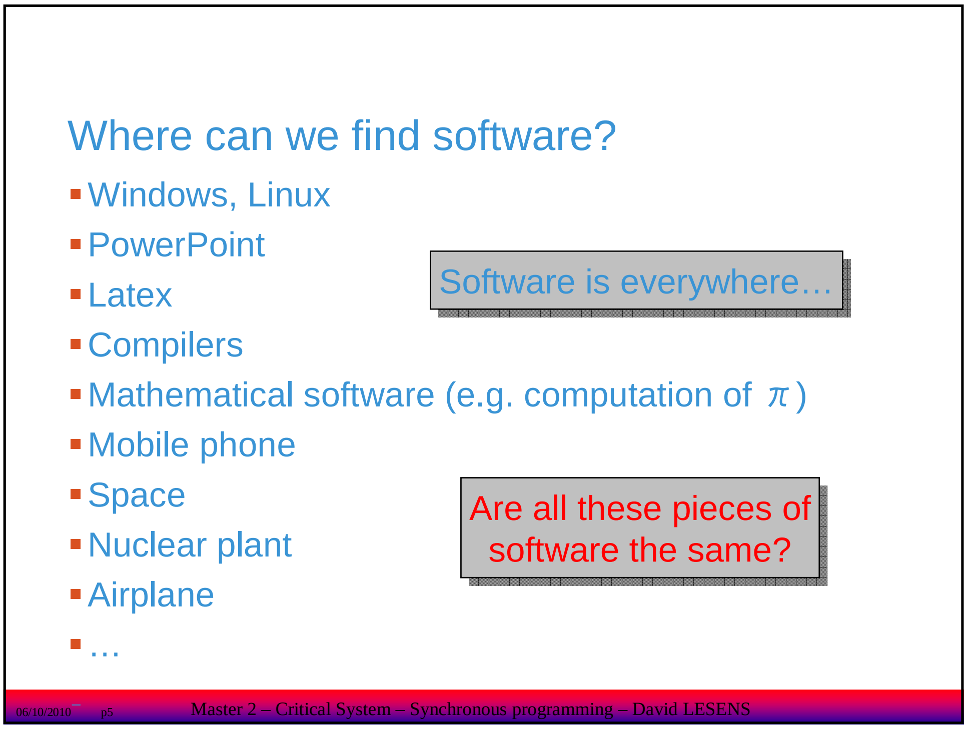
ASTRIUM AN EADS COMPANY



	MARDI	
8h		
8h30		
9h	Prog Synchrone CM/TD/TP E.Asarin, Lesens et M.Dogguy Script 443C	
9h30		
10h		
10h30		Pro ser (
11h		
11h30		
12h		
12h30		

Overview

- Critical real-time embedded software ☞
- Principles of the approach ☞
  - Introduction ☞
  - Formal semantics ☞
- SCADE ☞
- Model validation ☞

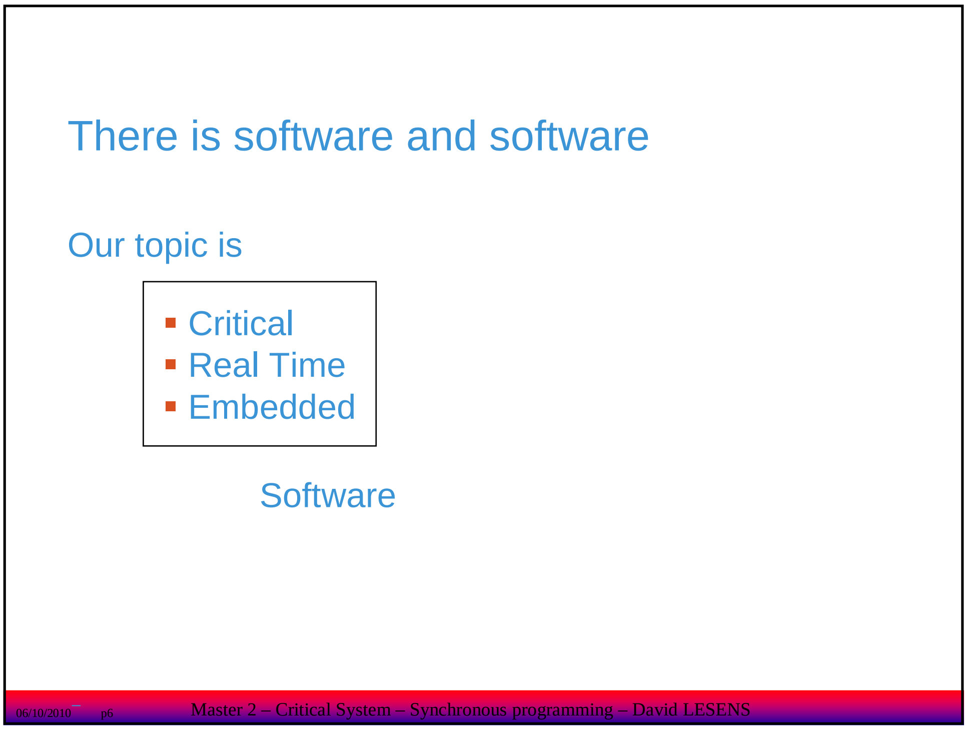


Where can we find software?

- Windows, Linux
- PowerPoint
- Latex
- Compilers
- Mathematical software (e.g. computation of π)
- Mobile phone
- Space
- Nuclear plant
- Airplane
- …

Software is everywhere…

Are all these pieces of software the same?



There is software and software

Our topic is

- Critical
- Real Time
- Embedded

Software

What is embedded software?

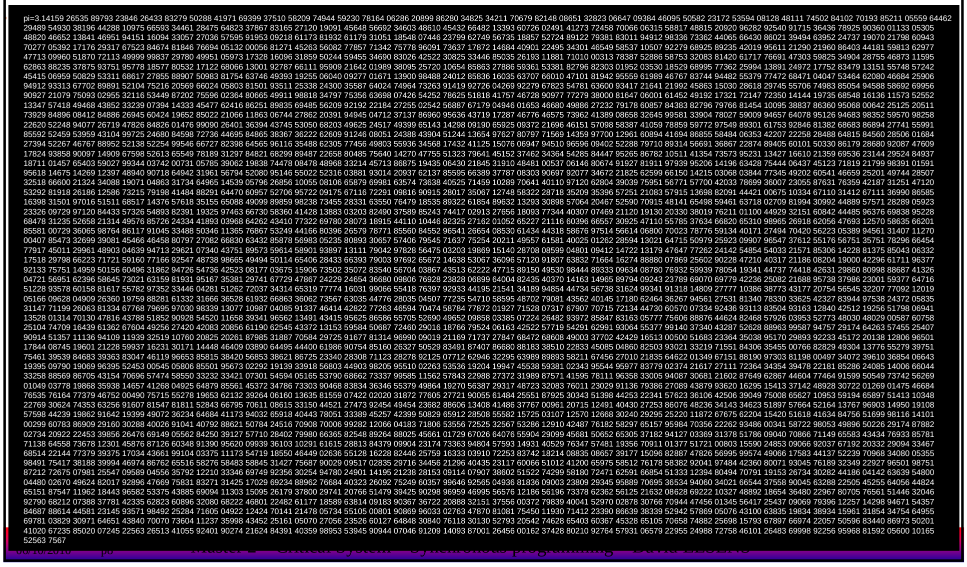
- Windows, Linux
- PowerPoint
- Latex
- Compilers
- Mathematical software (e.g. computation of π)

The software has its own objective
We can “buy” the software

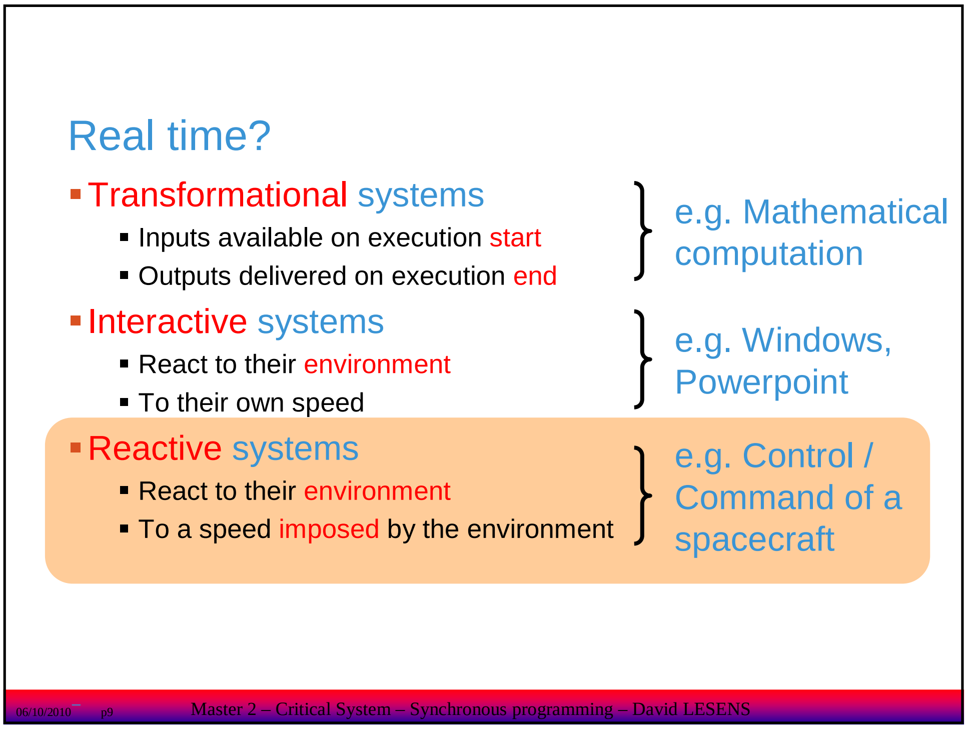
- Mobile phone
- Space launcher
- Nuclear plant
- Airplane

The software is part of the system
We can only “buy” the system

Compute the first 10,000 digits of Pi



```
pi=3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230 78164 06286 20899 86280 34825 34211 70679 82148 08651 32823 06647 09384 46095 50582 23172 53594 08128 48111 74502 84102 70193 85211 05559 64462
29489 54930 38196 44288 10975 66593 34461 28475 64823 37867 83165 27120 19091 45648 56692 34603 48610 45432 66482 13393 60726 02491 41273 72458 70066 06315 58817 48815 20920 96282 92540 91715 36436 78925 90360 01133 05305
48820 46652 13841 46951 94151 16094 33057 27036 57595 91953 09218 61173 81932 61179 31051 18548 07446 23799 62749 56735 18857 52724 89122 79381 83011 94912 98336 73362 44065 66430 86021 39494 63952 24737 19070 21798 60943
70277 05392 17176 29317 67523 84674 81846 76694 05132 00056 81271 45263 56082 77857 71342 75778 96091 73637 17872 14684 40901 22495 34301 46549 58537 10507 92279 68925 89235 42019 95611 21290 21960 86403 44181 59813 62977
47713 09960 51870 72113 49999 99837 29780 49951 05973 17328 16096 31859 50244 59455 34690 83026 42522 30825 33446 85035 26193 11881 71010 00313 78387 52886 58753 32083 81420 61717 76691 47303 59825 34904 28755 46873 11595
62863 88235 37875 93751 95778 18577 80532 17122 68066 13001 92787 66111 95909 21642 01989 38095 25720 10654 85863 27886 59361 53381 82796 82303 01952 03530 18529 68995 77362 25994 13891 24972 17752 83479 13151 55748 57242
45415 06959 50829 53311 68617 27855 88907 50983 81754 63746 49393 19255 06040 09277 01671 13900 98488 24012 85836 16035 63707 66010 47101 81942 95559 61989 46767 83744 94482 55379 77472 68471 04047 53464 62080 46684 25906
94912 93313 67702 89891 52104 75216 20569 66024 05803 81501 93511 25338 24300 35587 64024 74964 73263 91419 92726 04269 92279 67823 54781 63600 93417 21641 21992 45863 15030 28618 29745 55706 74983 85054 94588 58692 69956
90927 21079 75093 02955 32116 53449 87202 75596 02364 80665 49911 98818 34797 75356 63698 07426 54252 78625 51818 41757 46728 90977 77279 38000 81647 06001 61452 49192 17321 72147 72350 14144 19735 68548 16136 11573 52552
13347 57418 49468 43852 33239 07394 14333 45477 62416 86251 89835 69485 56209 92192 22184 27255 02542 56887 67179 04946 01653 46680 49886 27232 79178 60857 84383 82796 79766 81454 10095 38837 86360 95068 00642 25125 20511
73929 84896 08412 84886 26945 60424 19652 85022 21066 11863 06744 27862 20391 94945 04712 37137 86960 95636 43719 17287 46776 46575 73962 41389 08658 32645 99581 33904 78027 59009 94657 64078 95126 94683 98352 59570 98258
22620 52248 94077 26719 47826 84826 01476 99090 26401 36394 43745 53050 68203 49625 24517 49399 65143 14298 09190 65925 09372 21696 46151 57098 58387 41059 78859 59772 97549 89301 61753 92846 81382 68683 86894 27741 55991
85592 52459 53959 43104 99725 24680 84598 72736 44695 84865 38367 36222 62609 91246 08051 24388 43904 51244 13654 97627 80797 71569 14359 97700 12961 60894 41694 86855 58484 06353 42207 22258 28488 64815 84560 28506 01684
27394 52267 46767 88952 52138 52254 99546 66727 82398 64565 96116 35488 62305 77456 49803 55936 34568 17432 41125 15076 06947 94510 96596 09402 52288 79710 89314 56691 36867 22874 89405 60101 50330 86179 28680 92087 47609
17824 93858 90097 14909 67598 52613 65549 78189 31297 84821 68299 89487 22658 80485 75640 14270 47755 51323 79641 45152 37462 34364 54285 84447 95265 86782 10511 41354 73573 95231 13427 16610 21359 69536 23144 29524 84937
18711 01457 65403 59027 99344 03742 00731 05785 39062 19838 74478 08478 48968 33214 45713 86875 19435 06430 21845 31910 48481 00537 06146 80674 91927 81911 97939 95206 14196 63428 75444 06437 45123 71819 21799 98391 01591
95618 14675 14269 12397 48940 90718 64942 31961 56794 52080 95146 55022 52316 03881 93014 20937 62137 85595 66389 37787 08303 90697 92077 34672 21825 62599 66150 14215 03068 03844 77345 49202 60541 46659 25201 49744 28507
32518 66600 21324 34088 19071 04863 31734 64965 14539 05796 26856 10055 08106 65879 69981 63574 73638 40525 71459 10289 70641 40110 97120 62804 39039 75951 56771 57700 42033 78699 36007 23055 87631 76359 42187 31251 47120
53292 81918 26186 12586 73215 79198 41484 88291 64470 60957 52706 95722 09175 67116 72291 09816 90915 28017 35067 12748 58322 28718 35209 35396 57251 21083 57915 13698 82091 44421 00675 10334 67110 31412 67111 36990 86585
16398 31501 97016 51511 68517 14376 57618 35155 65088 49099 89859 98238 73455 28331 63550 76479 18535 89322 61854 89632 13293 30898 57064 20467 52590 70915 48141 65498 59461 63718 02709 81994 30992 44889 57571 28289 05923
23326 09729 97120 84433 57326 54893 82391 19325 97463 66730 58360 41428 13883 03203 82490 37589 85243 74417 02913 27656 18093 77344 40307 07469 21120 19130 20330 38019 76211 01100 44929 32151 60842 44485 96376 69838 95228
68478 31235 52658 21314 49576 85726 24334 41893 03968 64262 43410 77322 69780 28073 18915 44110 10446 82325 27162 01052 65227 21116 60396 66557 30925 47110 55785 37634 66820 65310 98965 26918 62056 47693 12570 58635 66201
85581 00729 36065 98764 86117 91045 33488 50346 11365 76867 53249 44166 80396 26579 78771 85560 84552 96541 26654 08530 61434 44318 58676 97514 56614 06800 70023 78776 59134 40171 27494 70420 56223 05389 94561 31407 11270
00407 85473 32699 39081 45466 46458 80797 27082 66830 63432 85878 56983 05235 80893 30657 57406 79545 71637 75254 20211 49557 61581 40025 01262 28594 13021 64715 50979 25923 09907 96547 37612 55176 56751 35751 78296 66454
77917 45011 29961 48903 04639 94713 29621 07340 43751 89573 59614 58901 93897 13111 79042 97828 56475 03203 19869 15140 28708 08599 04801 09412 14722 13179 47647 77262 24142 54854 54033 21571 85306 14228 81375 85043 06332
17518 29798 66223 71721 59160 77166 92547 48738 98665 49494 50114 65406 28433 66393 79003 97692 65672 14638 53067 36096 57120 91807 63832 71664 16274 88880 07869 25602 90228 47210 40317 21186 08204 19000 42296 61711 96377
92133 75751 14959 50156 60496 31862 94726 54736 42523 08177 03675 15906 73502 35072 83540 56704 03867 43513 62222 47715 89150 49530 98444 89333 09634 08780 76932 59939 78054 19341 44737 74418 42631 29860 80998 88687 41326
04721 56951 62396 58645 73021 63159 81931 95167 35381 29741 67729 47867 24229 24654 36680 09806 76928 23828 06899 64004 82435 40370 14163 14965 89794 09243 23789 69070 69779 42236 25082 21688 95738 37986 23001 59377 64716
51228 93578 60158 81617 55782 97352 33446 04281 51262 72037 34314 65319 77774 16031 99066 55418 76397 92933 44195 21541 34189 94854 44734 56738 31624 99341 91318 14809 27777 10386 38773 43177 20754 56545 32207 77092 12019
05166 09628 04909 26360 19759 88281 61332 31666 36528 61932 66863 36062 73567 63035 44776 28035 04507 77235 54710 58595 48702 79081 43562 40145 17180 62464 36267 94561 27531 81340 78330 33625 42327 83944 97538 24372 05835
31147 71199 26063 81334 67768 79695 97030 98339 13077 10987 04085 91337 46414 42822 77263 46594 70474 58784 77872 01927 71528 07317 67907 70715 72134 44730 60570 07334 92436 93113 83504 93163 12840 42512 19256 51798 06941
13528 01314 70130 47816 43788 51852 90928 54520 11658 39341 96562 13491 43415 95625 86586 55705 52690 49652 09858 03385 07224 26482 93972 85847 83163 05777 75606 88876 44624 82468 57926 03953 52773 48030 48029 00587 60758
25104 74709 16439 61362 67604 49256 27420 42083 20856 61190 62545 43372 13153 59584 50687 72460 29016 18766 79524 06163 42522 57719 54291 62991 93064 55377 99140 37340 43287 52628 88963 99587 94757 29174 64263 57455 25407
90914 51357 11136 94109 11939 32519 10760 20825 20261 87985 31887 70584 29725 91677 81314 96990 09019 21169 71737 27847 68472 68608 49003 37702 42429 16513 00500 51683 23364 35038 95170 29893 92233 45172 20138 12806 96501
17844 08745 19601 21228 59937 16231 30171 14448 46409 03890 64495 44400 61986 90754 85160 26327 50529 83491 87407 86680 88183 38510 22833 45085 04860 82503 93021 33219 71551 84306 35455 00766 82829 49304 13776 55279 39751
75461 39539 84683 39363 83047 46119 96653 85815 38420 56853 38621 86725 23340 28308 71123 28278 92125 07712 62946 32295 63989 89893 58211 67456 27010 21835 64622 01349 67151 88190 97303 81198 00497 34072 39610 36854 06643
19395 09790 19069 96395 52453 00545 05806 85501 95673 02292 19139 33918 56803 44903 98205 95510 02263 53536 19204 19947 45538 59381 02343 95544 95977 83779 02374 21617 27111 72364 34354 39478 22181 85286 24085 14006 66044
33258 88569 86705 43154 70696 57474 58550 33232 33421 07301 54594 05165 53790 68662 73337 99585 11562 57843 22988 27372 31989 87571 41595 78111 96358 33005 94087 30681 21602 87649 62867 44604 77464 91599 50549 73742 56269
01049 03778 19868 35938 14657 41268 04925 64879 85561 45372 34786 73303 90468 83834 36346 55379 49864 19270 56387 29317 48723 32083 76011 23029 91136 79386 27089 43879 93620 16295 15413 37142 48928 30722 01269 01475 46684
76535 76164 77379 46752 00490 75715 55278 19653 62132 39264 06160 13635 81559 07422 02020 31872 77605 27721 90055 61484 25551 87925 30343 51398 44253 22341 57623 36106 42506 39049 75008 65627 10953 59194 65897 51413 10348
22769 30624 74353 63256 91607 81547 81811 52843 66795 70611 08615 33150 44521 27473 92454 49454 23682 88606 13408 41486 37767 00961 20715 12491 40430 27253 86076 48236 34143 34623 51897 57664 52164 13767 96903 14950 19108
57598 44239 19862 91642 19399 49072 36234 64684 41173 94032 65918 40443 78051 33389 45257 42399 50829 65912 28508 55582 15725 03107 12570 12668 30240 29295 25220 11872 67675 62204 15420 51618 41634 84756 51699 98116 14101
00299 60783 86909 29160 30288 40026 91041 40792 88621 50784 24516 70908 70006 99282 12066 04183 71806 53556 72525 32567 53286 12910 42487 76182 58297 65157 95984 70356 22262 93486 00341 58722 98053 49896 50226 29174 87882
02734 20922 22453 39856 26476 69149 05562 84250 39127 57710 28402 79980 66365 82548 89264 88025 45661 01729 67026 64076 55904 29099 45681 50652 65305 37182 94127 03369 31378 51786 09040 70866 71149 65583 43434 76933 85781
71138 64558 73678 12301 45876 87126 60348 91390 95620 09939 36103 10291 61615 28813 84379 09904 23174 73363 94804 57593 14931 40529 76347 57481 19356 70911 01377 51721 00803 15590 24853 09066 92037 67192 20332 29094 33467
68514 22144 77379 39375 17034 43661 99104 03375 11173 54719 18550 46449 02636 55128 16228 82446 25759 16333 03910 72253 83742 18214 08835 08657 39177 15096 82887 47826 56995 99574 49066 17583 44137 52239 70968 34080 05355
98491 75417 38188 39994 46974 86762 65516 58276 58483 58845 31427 75687 90029 09517 02835 29716 34456 21296 40435 23117 60066 51012 41200 65975 58512 76178 58382 92041 97484 42360 80071 93045 76189 32349 22927 96501 98751
87212 72675 07981 25547 09589 04556 35792 12210 33346 69749 92356 30254 94780 24901 14195 21238 28153 09114 07907 38602 51522 74299 58180 72471 62591 66854 51333 12394 80494 70791 19153 26734 30282 44186 04142 63639 54800
04480 02670 49624 82017 92896 47669 75831 83271 31425 17029 69234 88962 76684 40323 26092 75249 60357 99646 92565 04936 81836 09003 23809 29345 95889 70695 36534 94060 34021 66544 37558 90045 63288 22505 45255 64056 44824
65151 87547 11962 18443 96582 53375 43885 69094 11303 15095 26179 37800 29741 20766 51479 39425 90298 96959 46995 56576 12186 56196 73378 62362 56125 21632 08628 69222 10327 48892 18654 36480 22967 80705 76561 51446 32046
92790 68212 07388 37781 42335 62823 60896 32080 68222 46801 22482 61177 18589 63814 09183 90367 36722 20888 32151 37556 00372 79839 40041 52970 02878 30766 70944 47456 01345 56417 25437 09069 79396 12257 14298 94671 54357
84687 88614 44581 23145 93571 98492 25284 71605 04922 12424 70141 21478 05734 55105 00801 90869 96033 02763 47870 81081 75450 11930 71412 23390 86639 38339 52942 57869 05076 43100 63835 19834 38934 15961 31854 34754 64955
69781 03829 30971 64651 43840 70070 73604 11237 35998 43452 25161 05070 27056 23526 60127 64848 30840 76118 30130 52793 20542 74628 65403 60367 45328 65105 70658 74882 25698 15793 67897 66974 22057 50596 83440 86973 50201
41020 67235 85020 07245 22563 26513 41055 92401 90274 21624 84391 40359 98953 53945 90944 07046 91209 14093 87001 26456 00162 37428 80210 92764 57931 06579 22955 24988 72758 46101 26483 69998 92256 95968 81592 05600 10165
52563 7567
```



Real time?

- Transformational systems
  - Inputs available on execution start
  - Outputs delivered on execution end

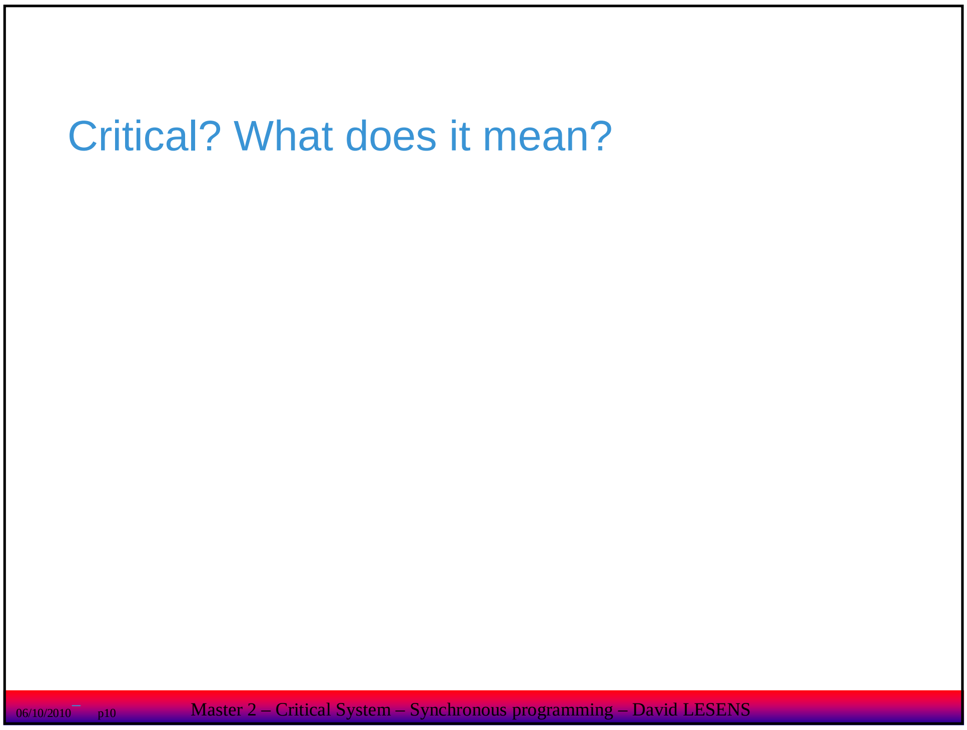
e.g. Mathematical computation

- Interactive systems
  - React to their environment
  - To their own speed

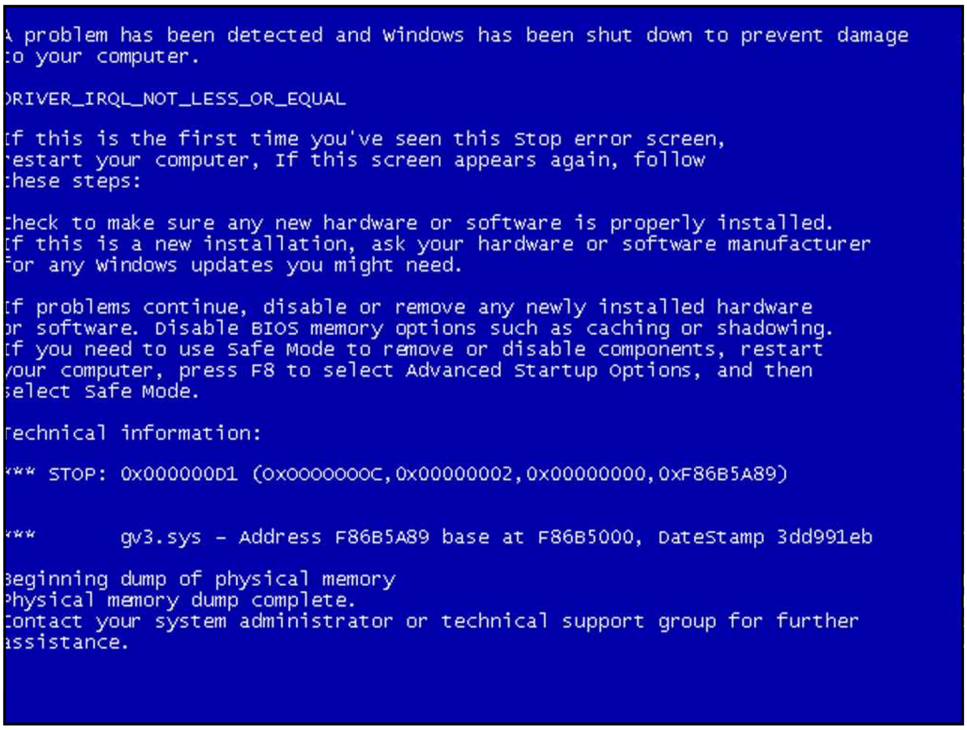
e.g. Windows, Powerpoint

- Reactive systems
  - React to their environment
  - To a speed imposed by the environment

e.g. Control / Command of a spacecraft



Critical? What does it mean?

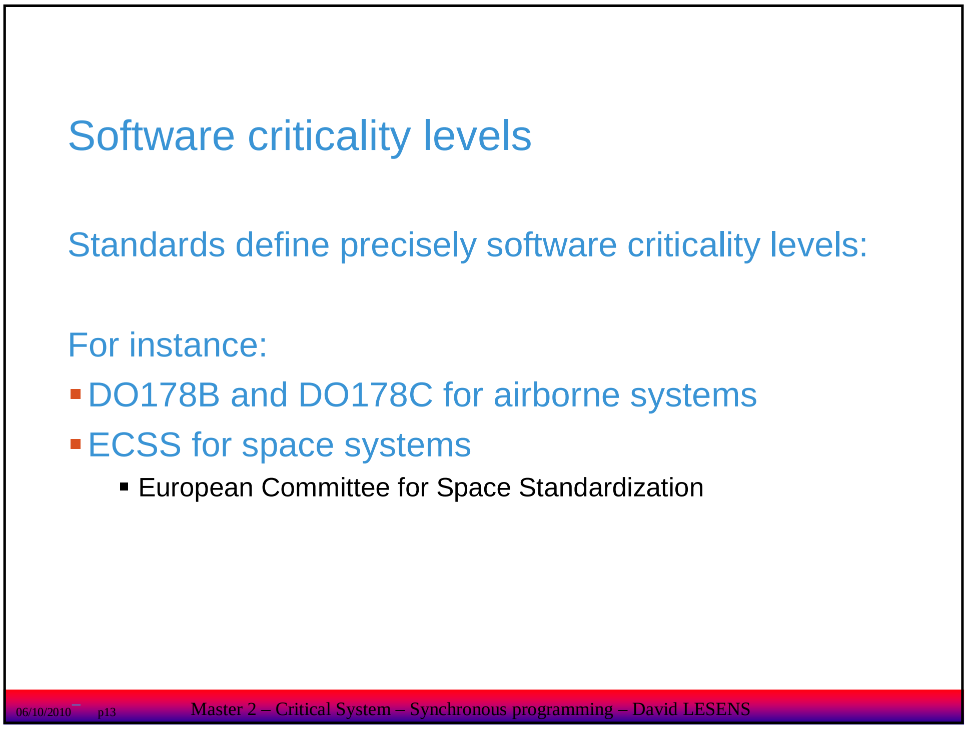


```
A problem has been detected and Windows has been shut down to prevent damage
to your computer.

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)


***       gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
```

Critical? What does it mean?

Intuitively, a critical system is a system which failure can have severe impacts

- Nuclear
- Aeronautic
- Automotive
- Railway
- Space
- …

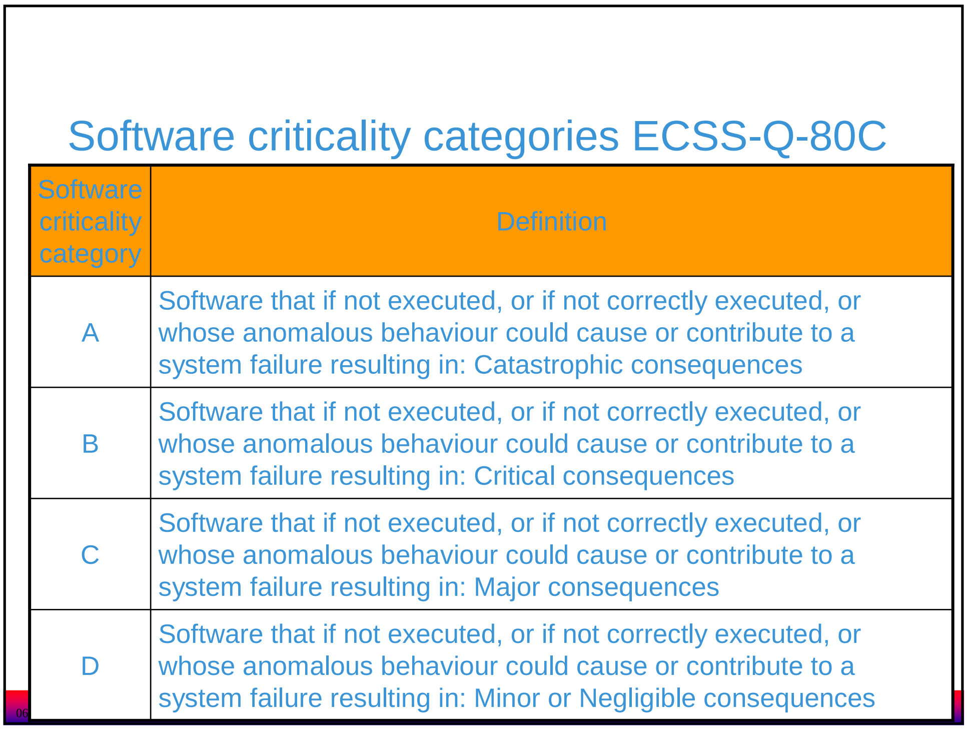


Software criticality levels

Standards define precisely software criticality levels:

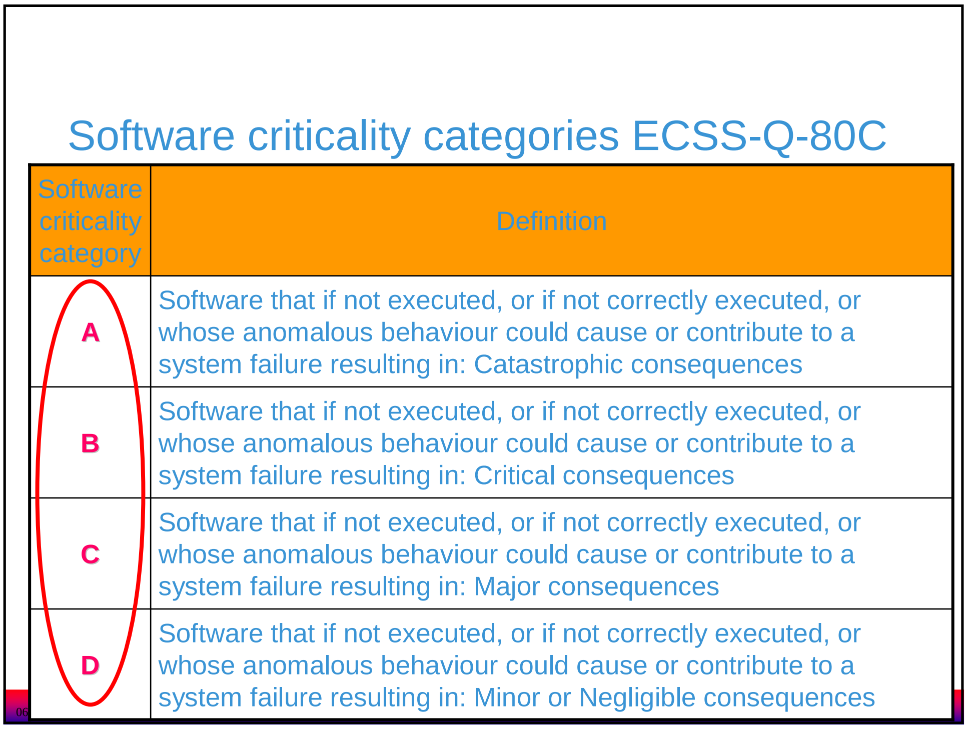
For instance:

- DO178B and DO178C for airborne systems
- ECSS for space systems
  - European Committee for Space Standardization



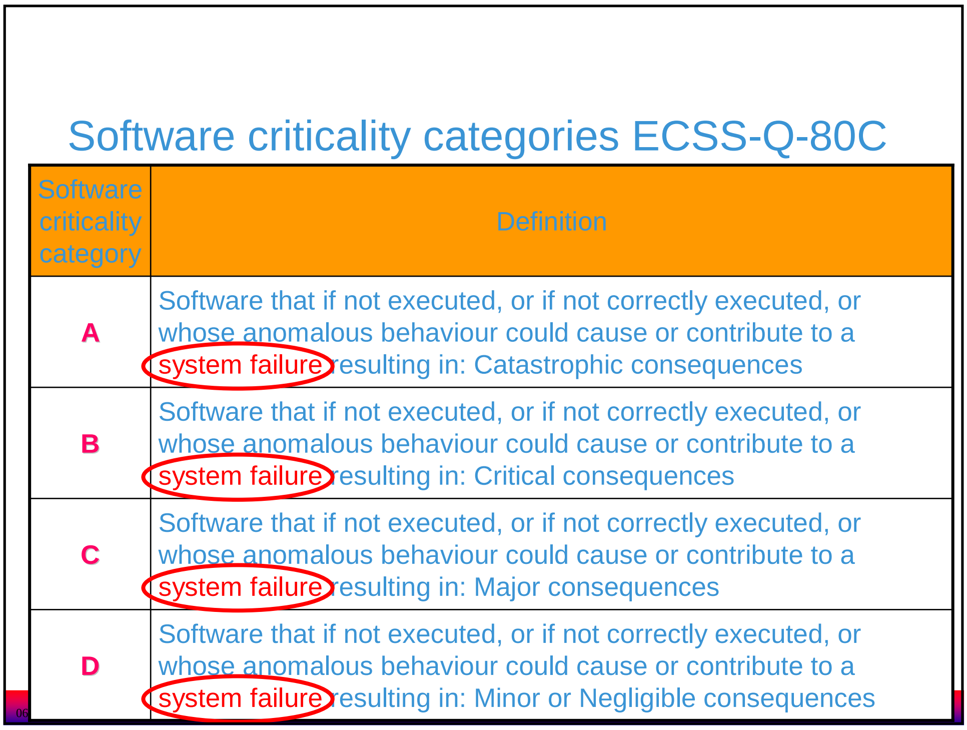
Software criticality categories ECSS-Q-80C

Software criticality category	Definition
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Catastrophic consequences
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Critical consequences
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Major consequences
D	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Minor or Negligible consequences



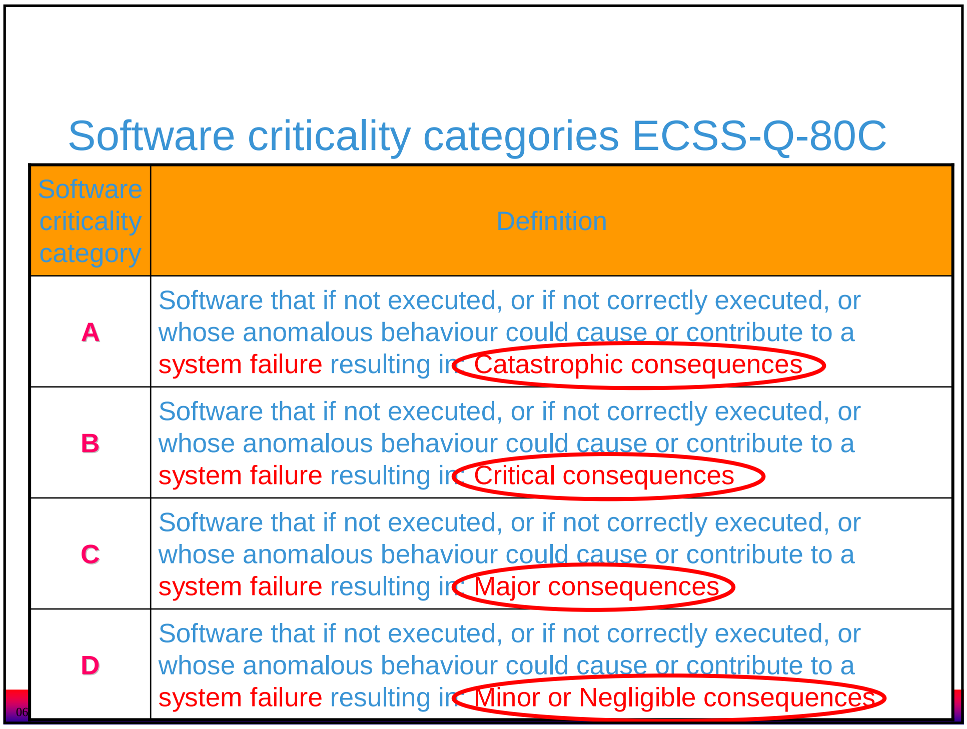
Software criticality categories ECSS-Q-80C

Software criticality category	Definition
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Catastrophic consequences
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Critical consequences
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Major consequences
D	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Minor or Negligible consequences



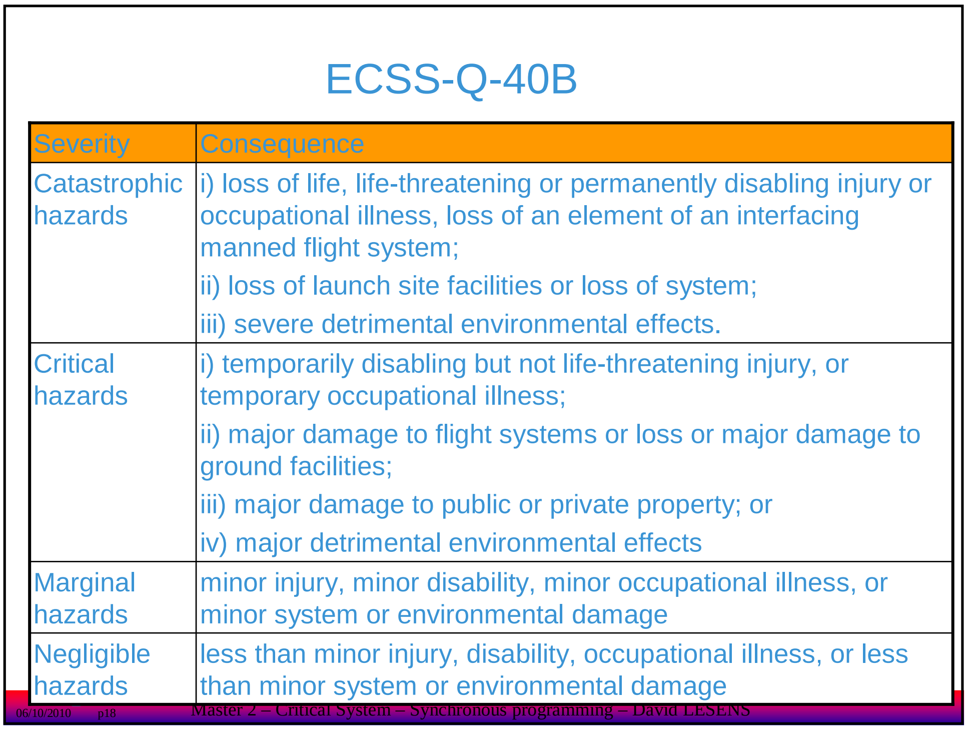
Software criticality categories ECSS-Q-80C

Software criticality category	Definition
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Catastrophic consequences
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Critical consequences
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Major consequences
D	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in: Minor or Negligible consequences



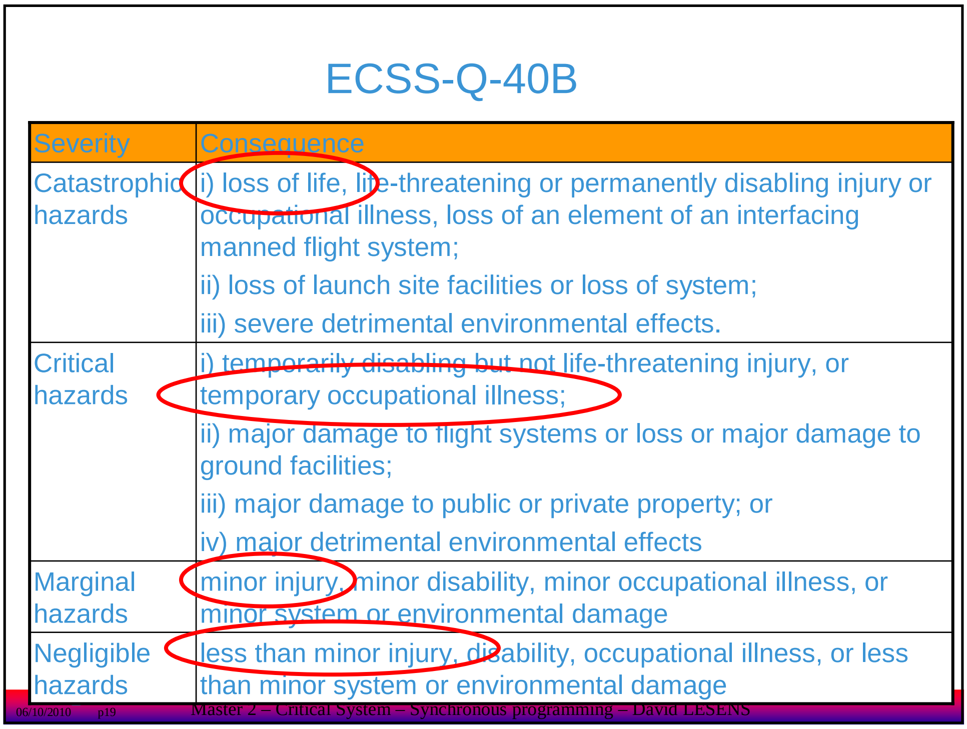
Software criticality categories ECSS-Q-80C

Software criticality category	Definition
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in Catastrophic consequences
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in Critical consequences
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in Major consequences
D	Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in Minor or Negligible consequences



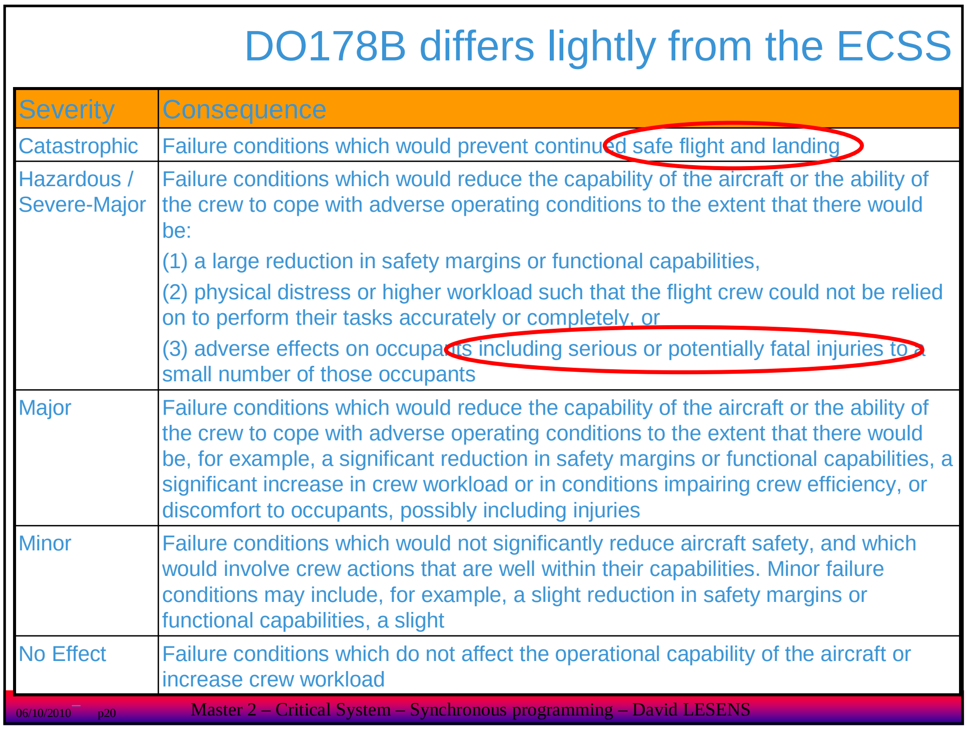
ECSS-Q-40B

Severity	Consequence
Catastrophic hazards	i) loss of life, life-threatening or permanently disabling injury or occupational illness, loss of an element of an interfacing manned flight system; ii) loss of launch site facilities or loss of system; iii) severe detrimental environmental effects.
Critical hazards	i) temporarily disabling but not life-threatening injury, or temporary occupational illness; ii) major damage to flight systems or loss or major damage to ground facilities; iii) major damage to public or private property; or iv) major detrimental environmental effects
Marginal hazards	minor injury, minor disability, minor occupational illness, or minor system or environmental damage
Negligible hazards	less than minor injury, disability, occupational illness, or less than minor system or environmental damage



ECSS-Q-40B

Severity	Consequence
Catastrophic hazards	i) loss of life, life-threatening or permanently disabling injury or occupational illness, loss of an element of an interfacing manned flight system; ii) loss of launch site facilities or loss of system; iii) severe detrimental environmental effects.
Critical hazards	i) temporarily disabling but not life-threatening injury, or temporary occupational illness; ii) major damage to flight systems or loss or major damage to ground facilities; iii) major damage to public or private property; or iv) major detrimental environmental effects
Marginal hazards	minor injury, minor disability, minor occupational illness, or minor system or environmental damage
Negligible hazards	less than minor injury, disability, occupational illness, or less than minor system or environmental damage



DO178B differs lightly from the ECSS

Severity	Consequence
Catastrophic	Failure conditions which would prevent continued safe flight and landing
Hazardous / Severe-Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be: (1) a large reduction in safety margins or functional capabilities, (2) physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or (3) adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants
Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to occupants, possibly including injuries
Minor	Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities. Minor failure conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight
No Effect	Failure conditions which do not affect the operational capability of the aircraft or increase crew workload

Wikipedia

Vocabulary

- Security
  - is the degree of protection against danger, loss, and criminals.
- Reliability
  - is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances.
- Safety
  - is the state of being "safe" (from French sauf), the condition of being protected against […] consequences of failure, damage, error, accidents, harm or any other event which could be considered non-desirable. It can include protection of people or of possessions.

Wikipedia

Safety & Security in Software Engineering

- The key difference between security and reliability is that security must take into account the actions of people attempting to cause destruction.

Safety

- The software must not harm the world

Security

- The world must not harm the software

Example 1: The First “Computer Bug”

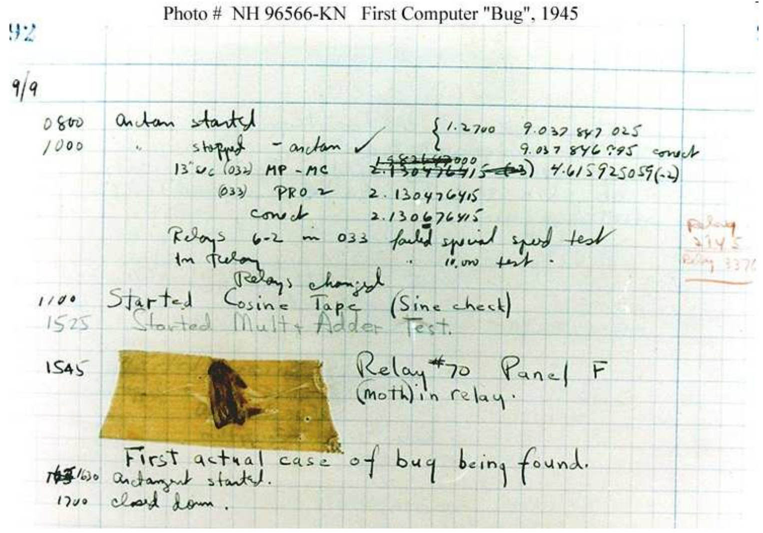


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025

9.037 846 795 correct

13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

in relay " 11,000 test .

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

~~1630~~ 1630 Antangent started.

1700 closed down.

Relay 2145

Relay 3370

Example 2: The Patriot Missile Failure

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people. A report of the General Accounting office, GAO/IMTEC-92-26, entitled *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia* reported on the cause of the failure. It turns out that the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors.

Failure in space

Trident

Sea launch

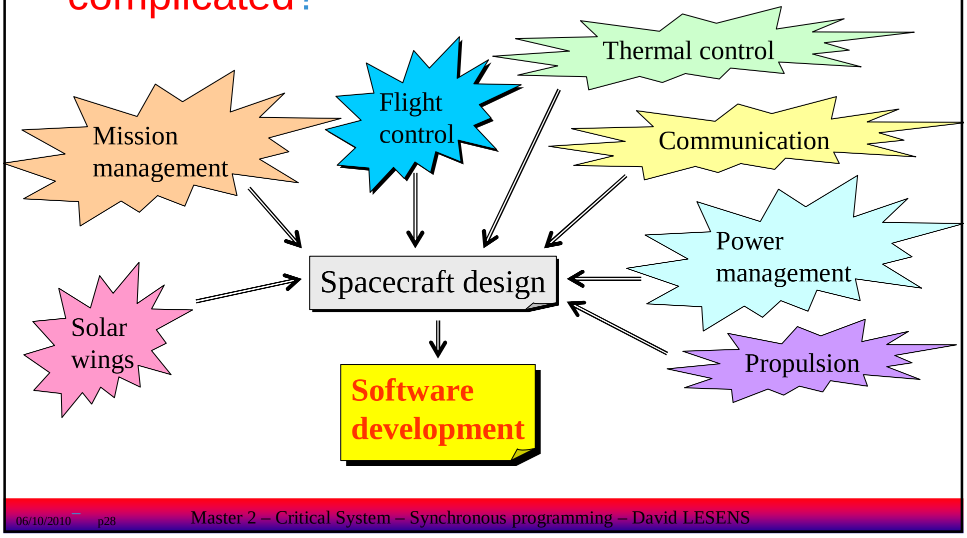
Overview

- Critical real-time embedded software ☞
- Principles of the approach ☞
  - Introduction ☞
  - Formal semantics ☞
- SCADE ☞
- Model validation ☞

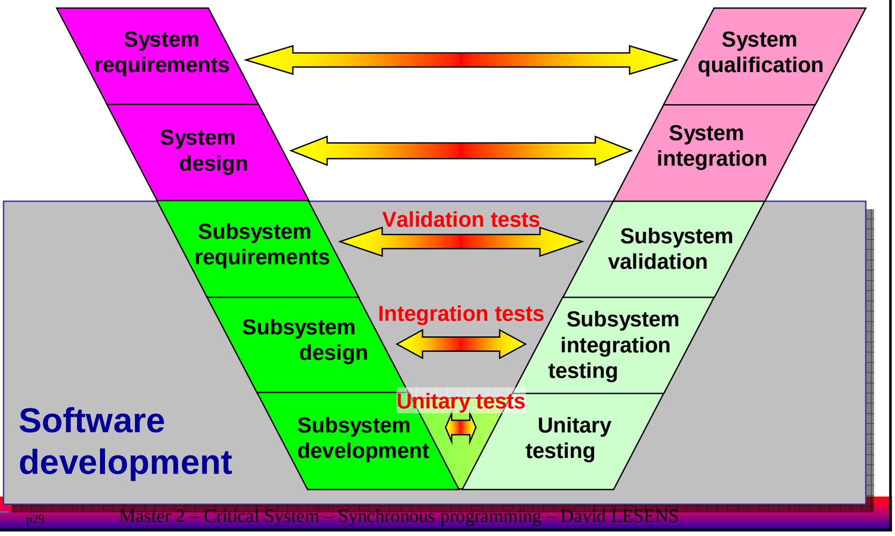
NASA's Climate Orbiter was lost September 23, 1999, due to a software bug

One engineering team used metric units while another used English units

Why is System (to Software) Engineering complicated?



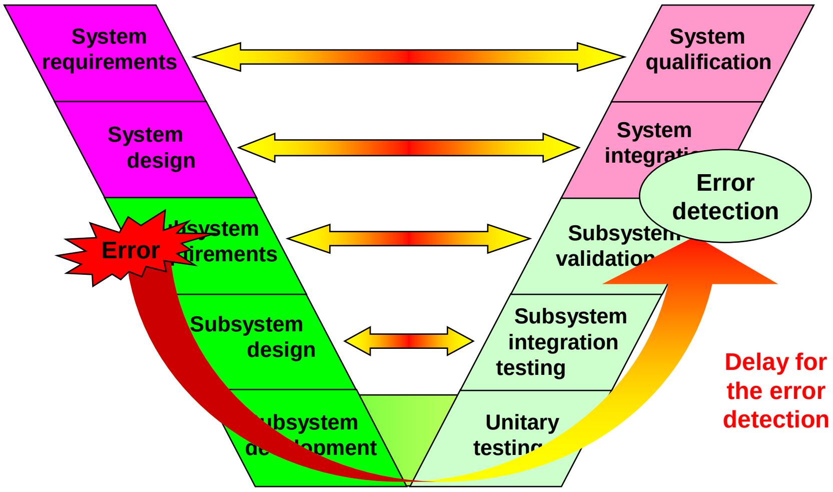
The V development cycle



Costs of critical software development

Specification	10%
Design	10%
Development/TU	25%
Integration tests	5%
Validation	**50%**

Late detection of errors



System requirements

System design

Subsystem requirements

Subsystem design

Subsystem development

System qualification

System integration

Subsystem validation

Subsystem integration testing

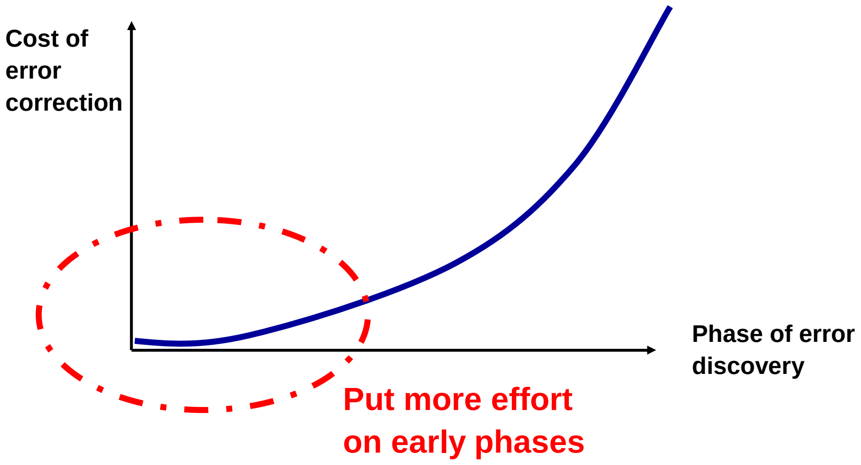
Unitary testing

Error

Error detection

Delay for the error detection

Cost of error correction

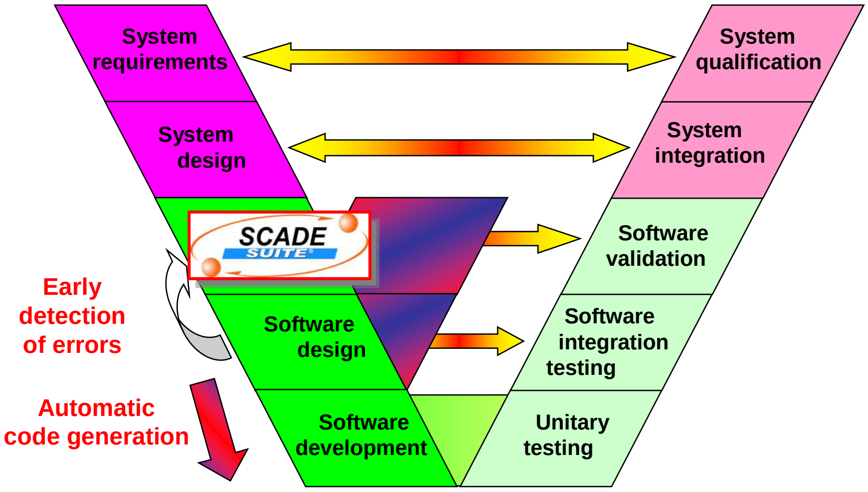


Cost of error correction

Phase of error discovery

Put more effort on early phases

Verification with model driven engineering

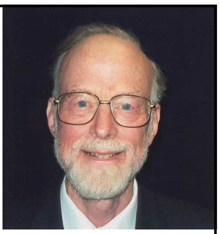


Formal Model Driven Engineering shall allow

- An early verification of the specification via a strong and intuitive semantic ensuring
  - Consistency
  - Completeness
  - Non ambiguity
- A behavioural validation within a simulation environment
- Automatic generation of certified code
- Formal proof

Overview

- Critical real-time embedded software ☞
- Principles of the approach ☞
  - Introduction ☞
  - Formal semantics ☞
- SCADE ☞
- Model validation ☞



There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

Professor C. A. R. Hoare
The 1980 Turing award lecture

Wikipedia

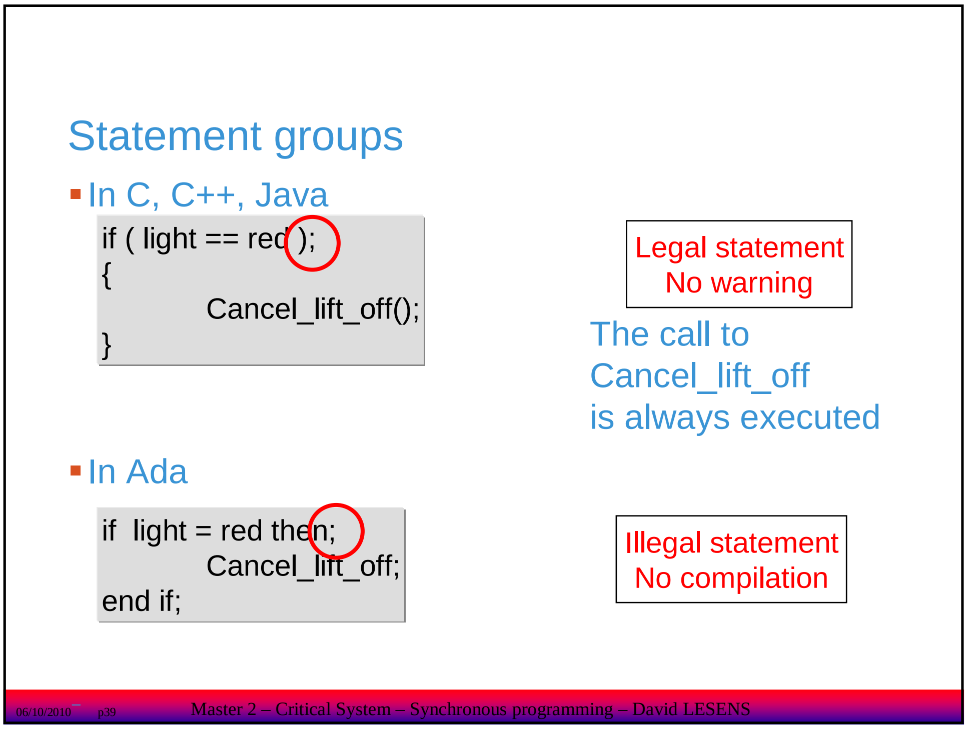
Formal semantics of programming languages

In theoretical computer science, formal semantics is the field concerned with the rigorous mathematical study of the meaning of programming languages and models of computation

Syntax

- Is it only what you say that matters?
- And not so much how it is said?

- A good syntax shall be
  - Clear
  - Unambiguous
  - Intuitive



Statement groups

- In C, C++, Java

```
if ( light == red );
{
            Cancel_lift_off();
}
```

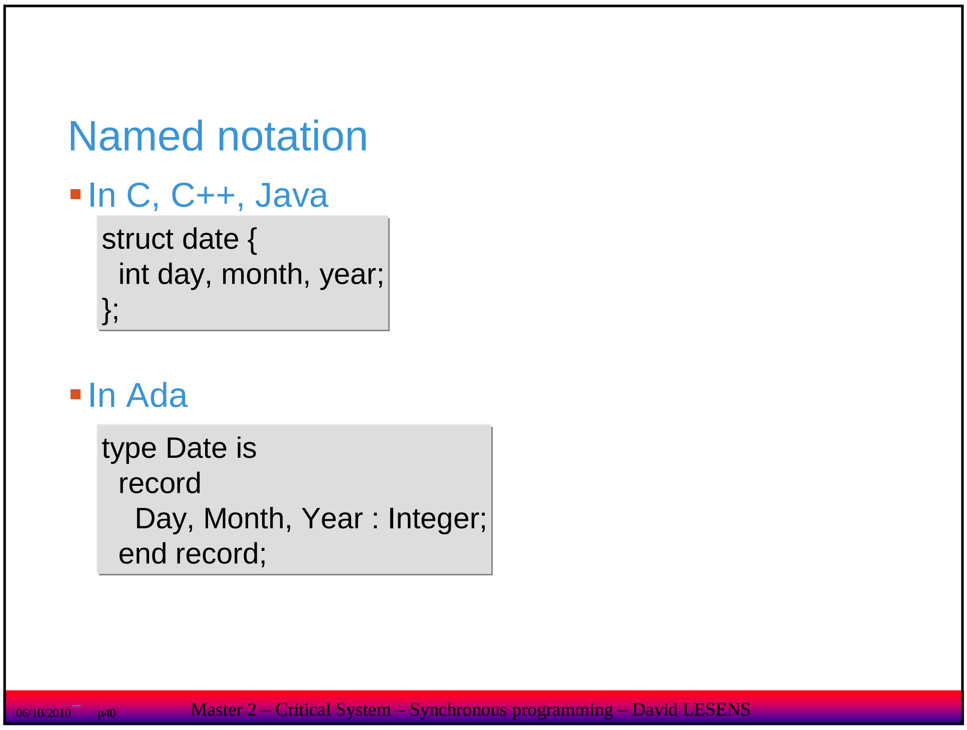
Legal statement
No warning

The call to
Cancel_lift_off
is always executed

- In Ada

```
if  light = red then;
            Cancel_lift_off;
end if;
```

Illegal statement
No compilation



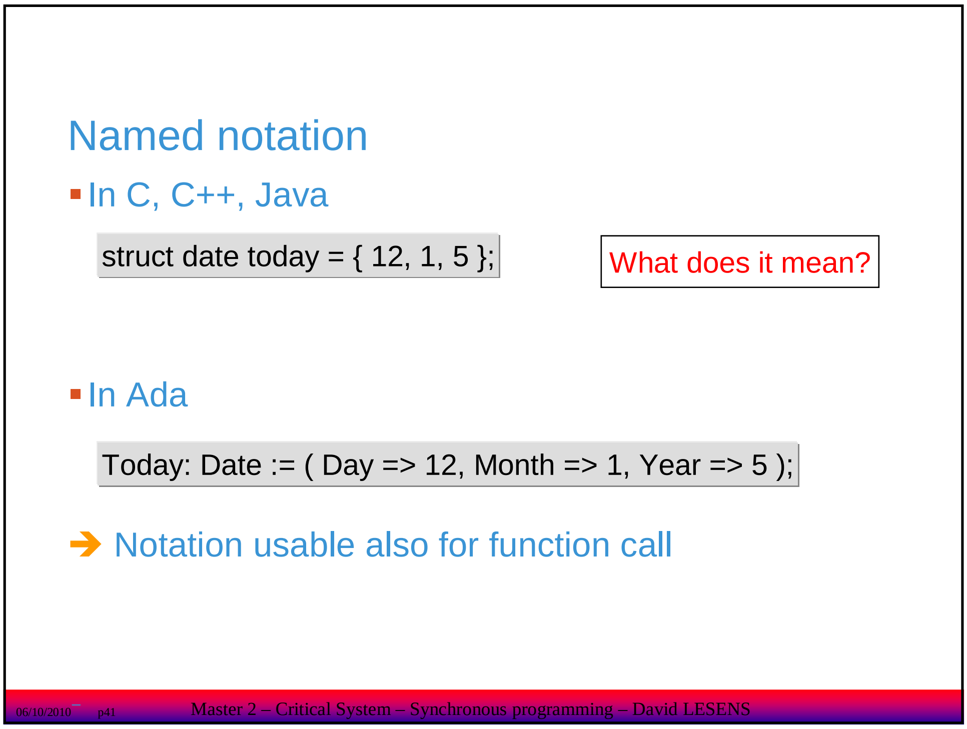
Named notation

- In C, C++, Java

```
struct date {
  int day, month, year;
};
```

- In Ada

```
type Date is
  record
    Day, Month, Year : Integer;
  end record;
```



Named notation

- In C, C++, Java

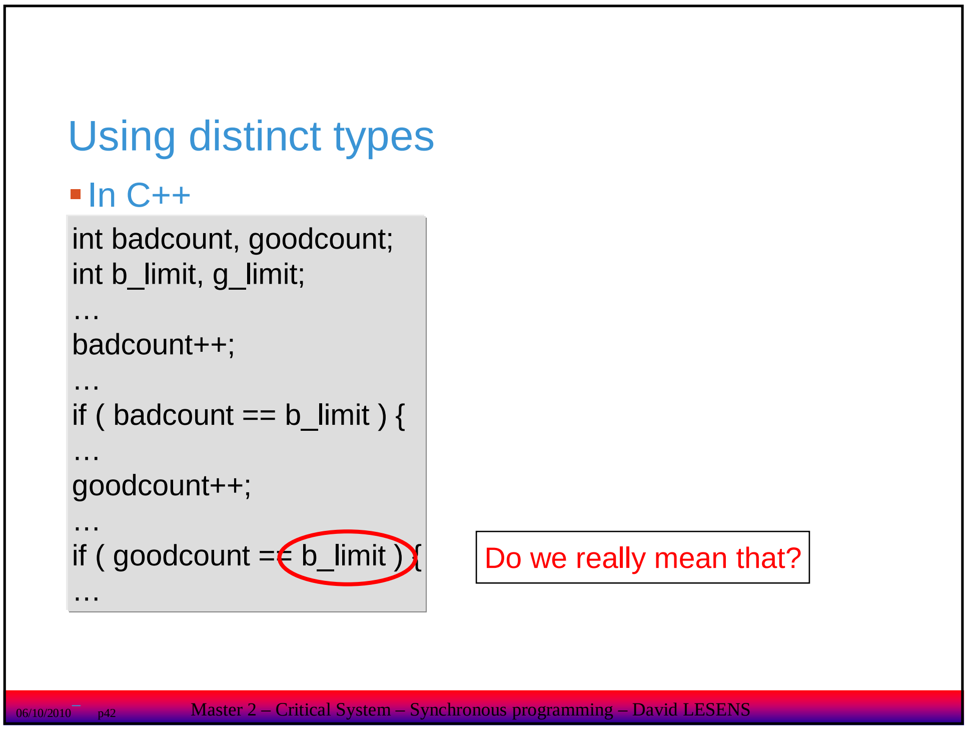
```
struct date today = { 12, 1, 5 };
```

What does it mean?

- In Ada

```
Today: Date := ( Day => 12, Month => 1, Year => 5 );
```

➔ Notation usable also for function call

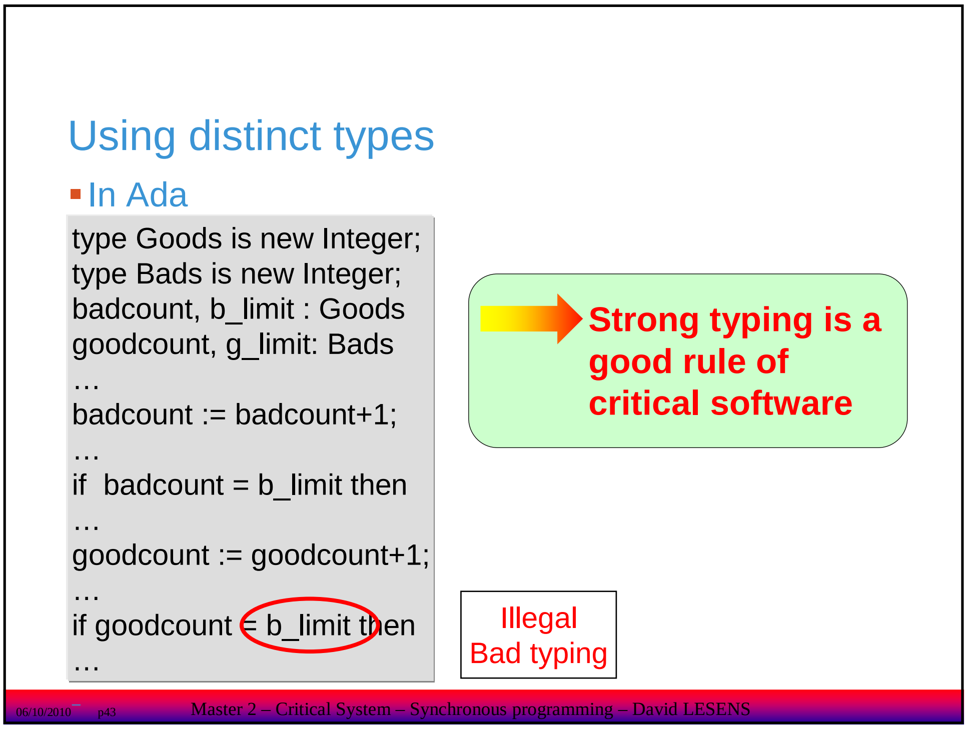


Using distinct types

- In C++

```
int badcount, goodcount;
int b_limit, g_limit;
…
badcount++;
…
if ( badcount == b_limit ) {
…
goodcount++;
…
if ( goodcount == b_limit ) {
…
```

Do we really mean that?



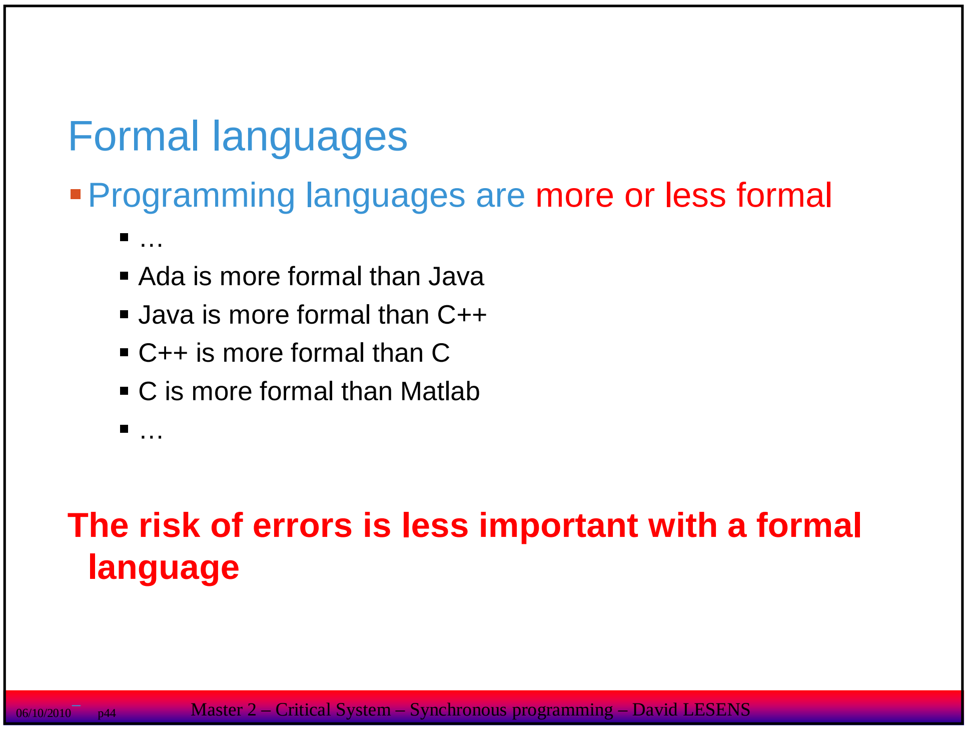
Using distinct types

- In Ada

```
type Goods is new Integer;
type Bads is new Integer;
badcount, b_limit : Goods
goodcount, g_limit: Bads
…
badcount := badcount+1;
…
if  badcount = b_limit then
…
goodcount := goodcount+1;
…
if goodcount = b_limit then
…
```

Illegal
Bad typing

Strong typing is a good rule of critical software



Formal languages

- Programming languages are more or less formal
  - …
  - Ada is more formal than Java
  - Java is more formal than C++
  - C++ is more formal than C
  - C is more formal than Matlab
  - …

The risk of errors is less important with a formal language

```
case State is
 when State1 =>
  Guard1 := X < 3;
  Guard2 := X > 3;
  if (EVENT1 and (Guard1 or Guard2)) then
    if (Guard1) then
     X := 5;
     State := State2;
    else
     if (Guard2) then
       X := 6;
     end if;
     State := State3;
    end if;
  end if;
 when State2 =>
  if (EVENT1) then
    X := 7;
    State := State1;
  end if;
 when State3 =>
  if (EVENT1 and EVENT1) then
    X := 8;
    State := State1;
  end if;
end case;
```

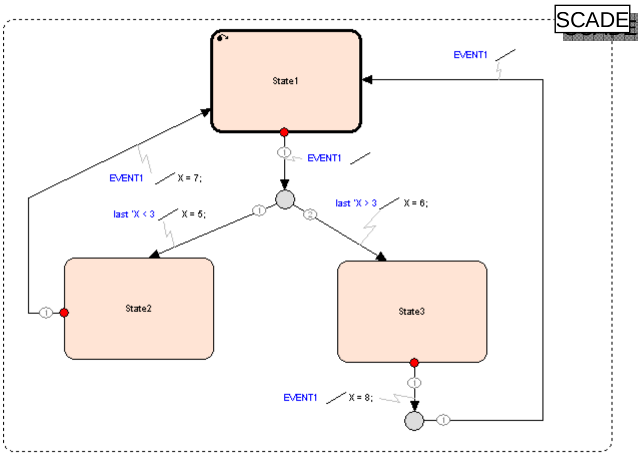
An other very simple example

Simple? Yes…

But what does this piece of code do?

Code (even Ada) is not an adequate way to communicate with system engineer

The same very simple example



➔ A graphical language with a high level of abstraction facilitates the communication

Overview

- Critical real-time embedded software
- Principles of the approach
  - Introduction
  - Formal semantics
- SCADE
- Model validation

Overview

- Synchronous model
- Introduction to the Scade language
- Editing a Scade model
- Activation conditions
- Automata
- Arrays
- Iterations
- Global flows: Sensors and probes
- Genericity

Wikipedia

Need of deterministic algorithm

- In computer science, a deterministic algorithm is an algorithm which, in informal terms, behaves predictably
- Given a particular input, it will always produce the same output, and the underlying machine will always pass through the same sequence of states

Wikipedia

Determinism and ECSS

ECSS-Q-80C

- 6.2.3 Handling of critical software
- 6.2.3.2 The supplier shall define and apply measures to assure the dependability and safety of critical software. These measures can include:
  - …
  - prohibiting the use of language commands and features that are unpredictable;
  - use of **formal design language for formal proof**;

Synchronous languages

Semantics = synchronous hypothesis

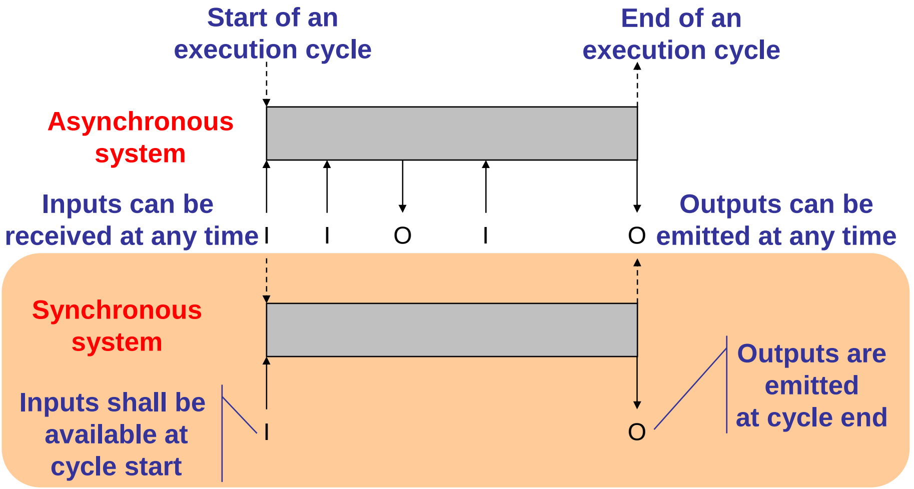
- Existence of a global clock
  - Software cyclically activated
  - Inputs read at the cycle beginning
  - Outputs delivered at cycle end

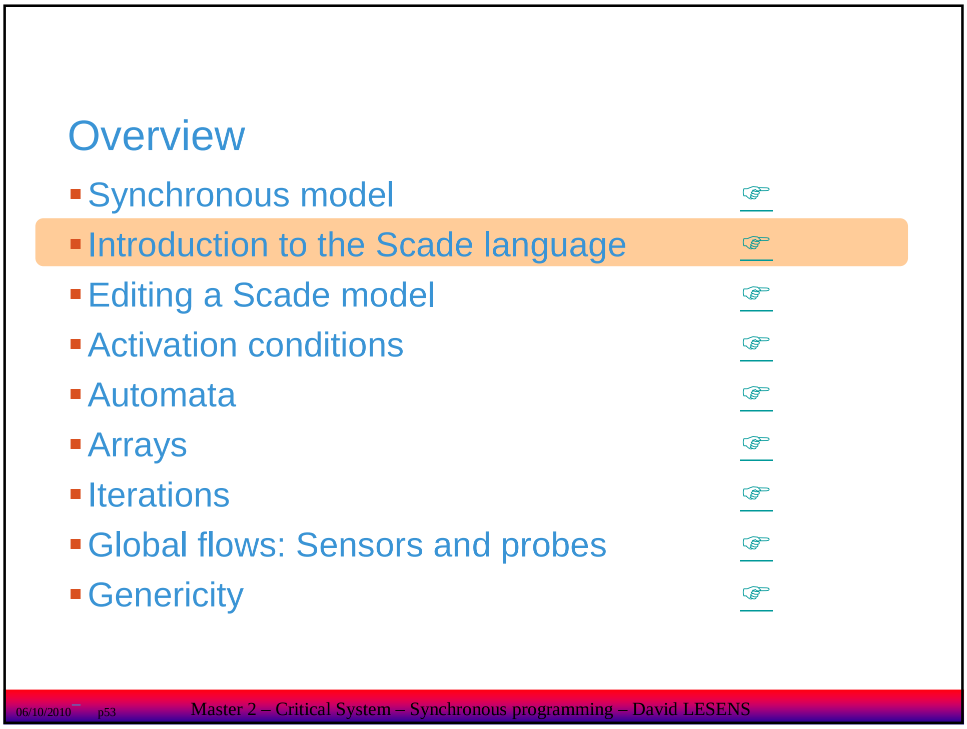
    (read / write forbidden during the cycle)
- The cycle execution duration shall theoretically be null

  ➔ No cycle overflow
- Mono-tasking

➔ Ensures the determinism

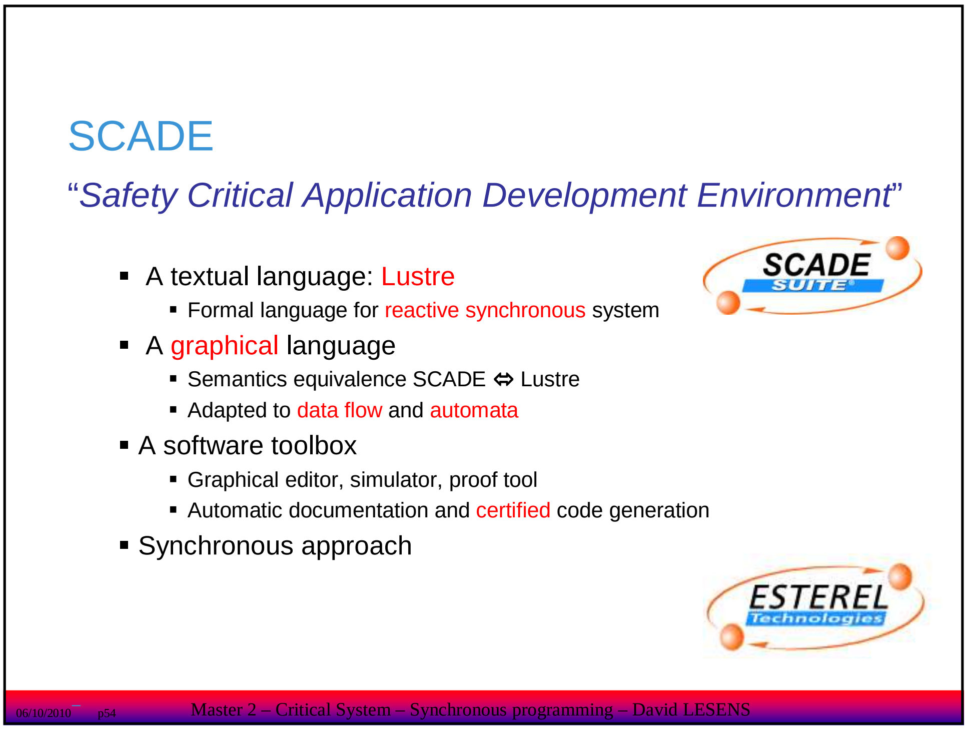
Asynchronous versus synchronous


Start of an execution cycle
End of an execution cycle
Asynchronous system
Inputs can be received at any time
I
I
O
I
O
Outputs can be emitted at any time
Synchronous system
Inputs shall be available at cycle start
I
O
Outputs are emitted at cycle end



Overview

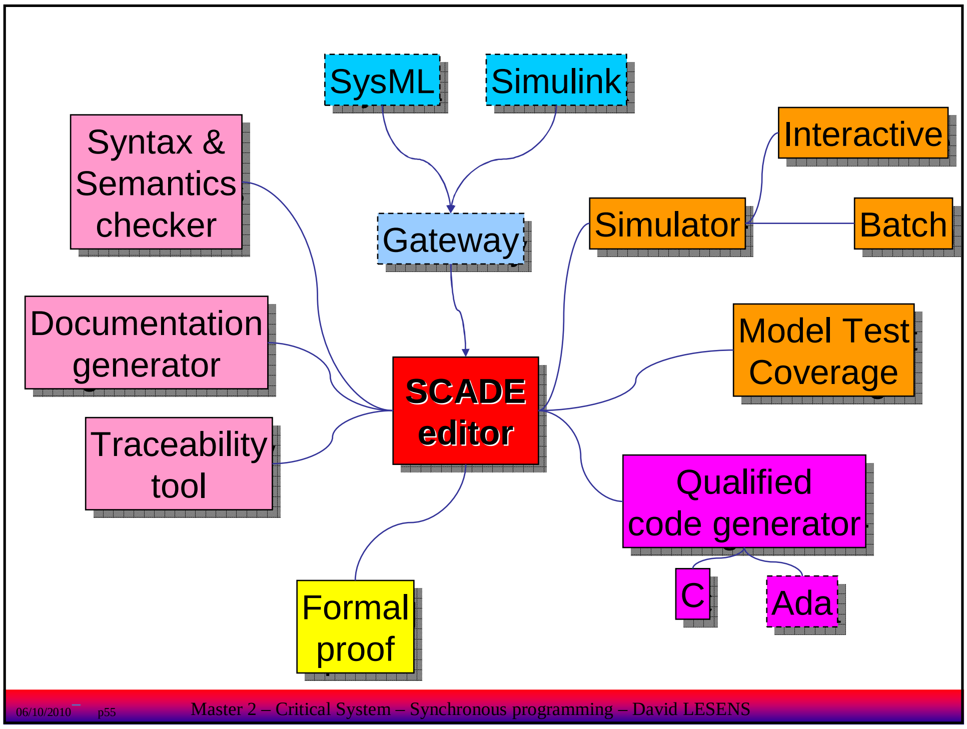
- Synchronous model ☞
- Introduction to the Scade language ☞
- Editing a Scade model ☞
- Activation conditions ☞
- Automata ☞
- Arrays ☞
- Iterations ☞
- Global flows: Sensors and probes ☞
- Genericity ☞



SCADE

"*Safety Critical Application Development Environment*"

- A textual language: Lustre
  - Formal language for reactive synchronous system
- A graphical language
  - Semantics equivalence SCADE ⇔ Lustre
  - Adapted to data flow and automata
- A software toolbox
  - Graphical editor, simulator, proof tool
  - Automatic documentation and certified code generation
- Synchronous approach



SCADE editor

SysML

Simulink

Gateway

Syntax & Semantics checker

Documentation generator

Traceability tool

Formal proof

Simulator

Interactive

Batch

Model Test Coverage

Qualified code generator

C

Ada

Time in Scade

- Global clock (known by all processes)
  - Time = discrete sequence of tick t_0, t_1, t_2, etc.
  - At each tick t_i a cycle is running
- Variable = flow which takes at each tick a unique value

  Example: integer variable x

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5

Operators

- An operator acts on flows of values (and not on values)

Example

- Operator « + »: int_n x int_n → int_n

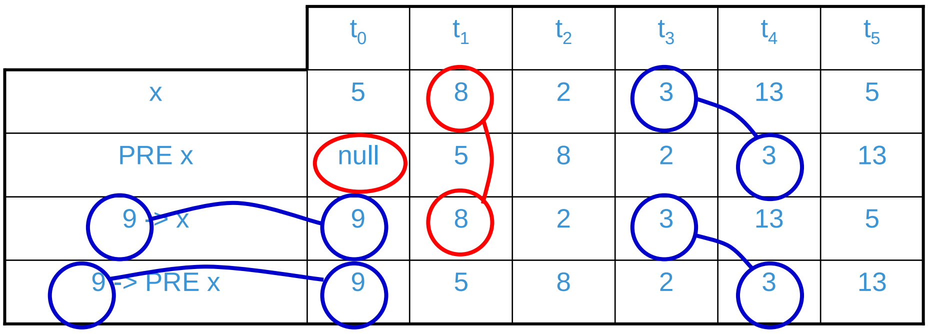
	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
x + x	10	16	4	6	26	10

Temporal operators

- The “PRE” operator takes as input a data flow (i.e. a variable) and returns its value at the previous tick.

  At initial tick, its value is undefined.
- The “→” operator takes as input an initialisation value and a data flow of the same type. It returns an identical data flow, except for the initial value.

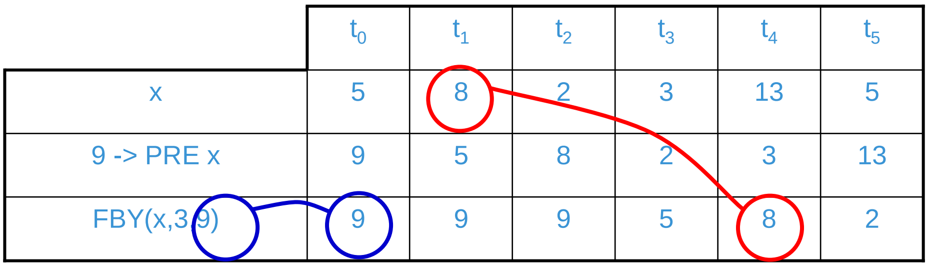
Example



	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
PRE x	null	5	8	2	3	13
9 -> x	9	8	2	3	13	5
9 -> PRE x	9	5	8	2	3	13

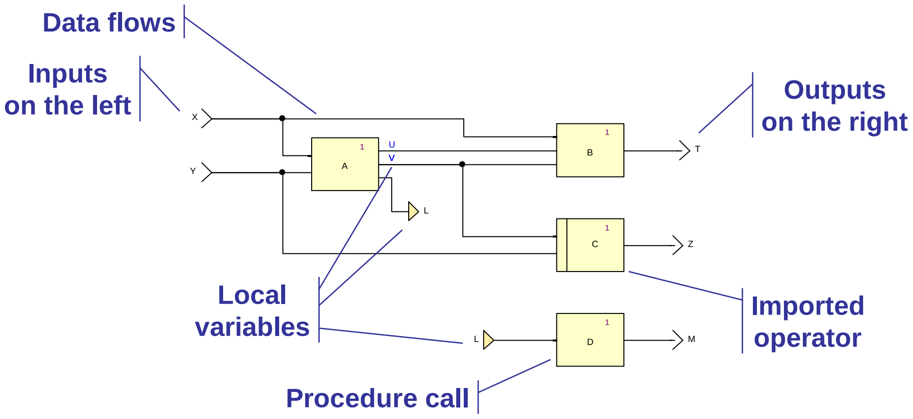
“Follow by” operator

$$\mathrm{FBY}(\ x,\ n,\ init\ ) = init \rightarrow \underbrace{(\ \mathrm{PRE}\ (\ \mathrm{PRE}\ \ldots\ x\ )\ )}_{n\ \text{times}}$$



	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
9 -> PRE x	9	5	8	2	3	13
FBY(x,3,9)	9	9	9	5	8	2

SCADE at a glance: Data Flow



Data flows

Inputs on the left

Outputs on the right

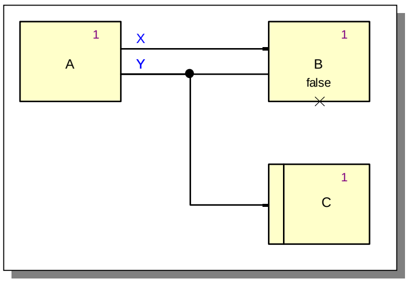
Local variables

Imported operator

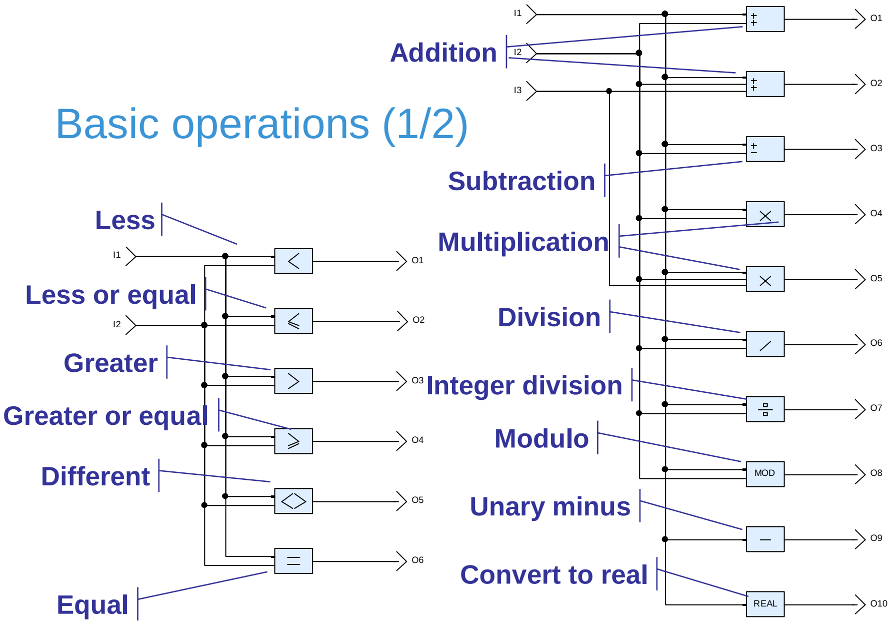
Procedure call

Textual versus graphical

```
( x, y ) = A();
B( x, y );
C( y )
```



Basic operations (1/2)


Addition

Subtraction

Multiplication

Division

Integer division

Modulo

Unary minus

Convert to real

Less

Less or equal

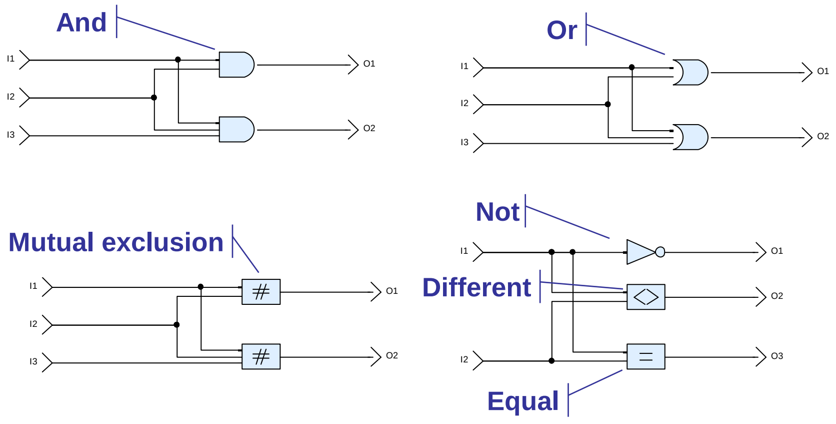
Greater

Greater or equal

Different

Equal

Basic operations (2/2)


And

Or

Mutual exclusion

Not

Different

Equal

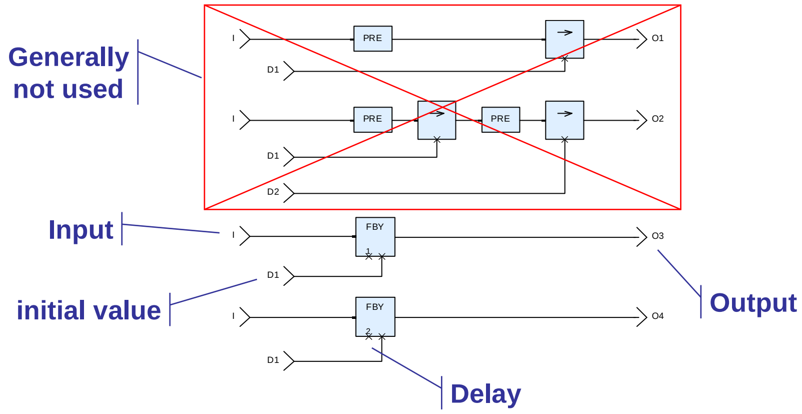
“Mutual exclusion” operator

$$\#: \underbrace{\text{bool} \times \text{bool} \times \ldots \times \text{bool}}_{\text{n times}} \rightarrow \text{bool}$$

Returns true if <u>at most</u> one of its inputs is true

e1	e2	e3	#(e1, e2, e3)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Delays



Generally not used

Input

initial value

Output

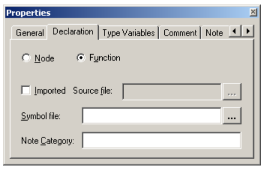
Delay

Node and function

y = f(x)

Function and nodes are represented by a rectangle

- A node has an internal state
- A function has **no** internal state



Properties

○ Node ● Function

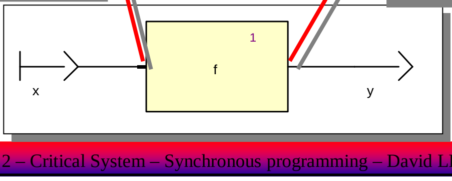
□ Imported Source file:

Symbol file:

Note Category:

Input parameters on the left

Output parameters on the right



Imported function / node

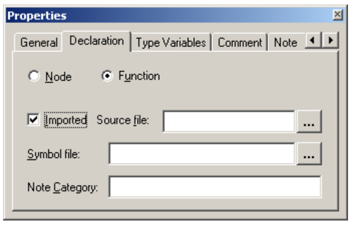
- Imported function

```
extern void C(
         bool Y );
```

- Imported node

```
extern void C_reset(
         outC_C *outC );
extern void C(
         bool Y,
         outC_C *outC );
```

Context to be defined by the developer



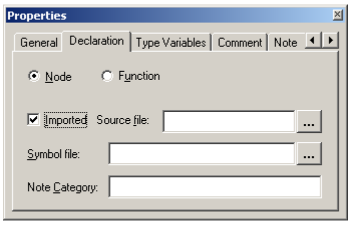
Properties

○ Node ● Function

☑ Imported Source file:

Symbol file:

Note Category:



Properties

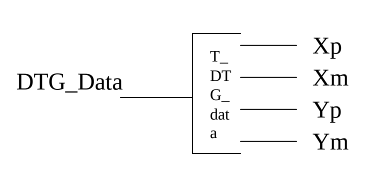
● Node ○ Function

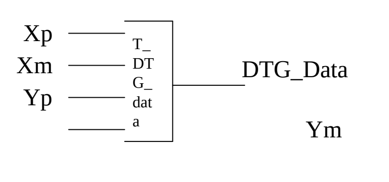
☑ Imported Source file:

Symbol file:

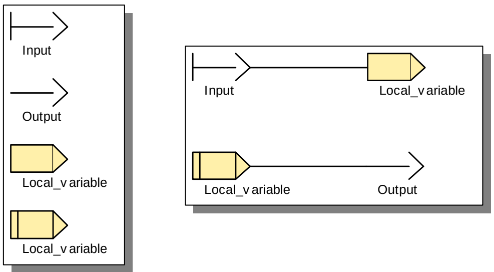
Note Category:

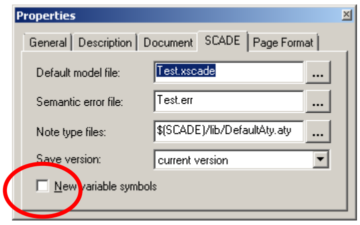
Data structure

	Xp := DTG_data.Xp
	Xm := DTG_data.Xm
	Yp := DTG_data.Yp
	Ym := DTG_data.Ym

	DTG_data.Xp := Xp
	DTG_data.Xm := Xm
	DTG_data.Yp := Yp
	DTG_data.Ym := Ym

Variables representation





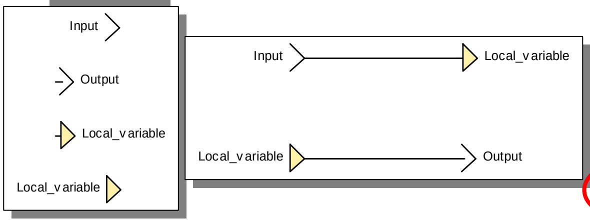
Default model file: Test.xscade

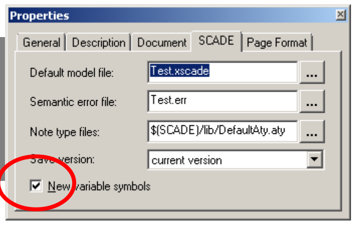
Semantic error file: Test.err

Note type files: $(SCADE)/lib/DefaultAty.aty

Save version: current version

□ New variable symbols





Default model file: Test.xscade

Semantic error file: Test.err

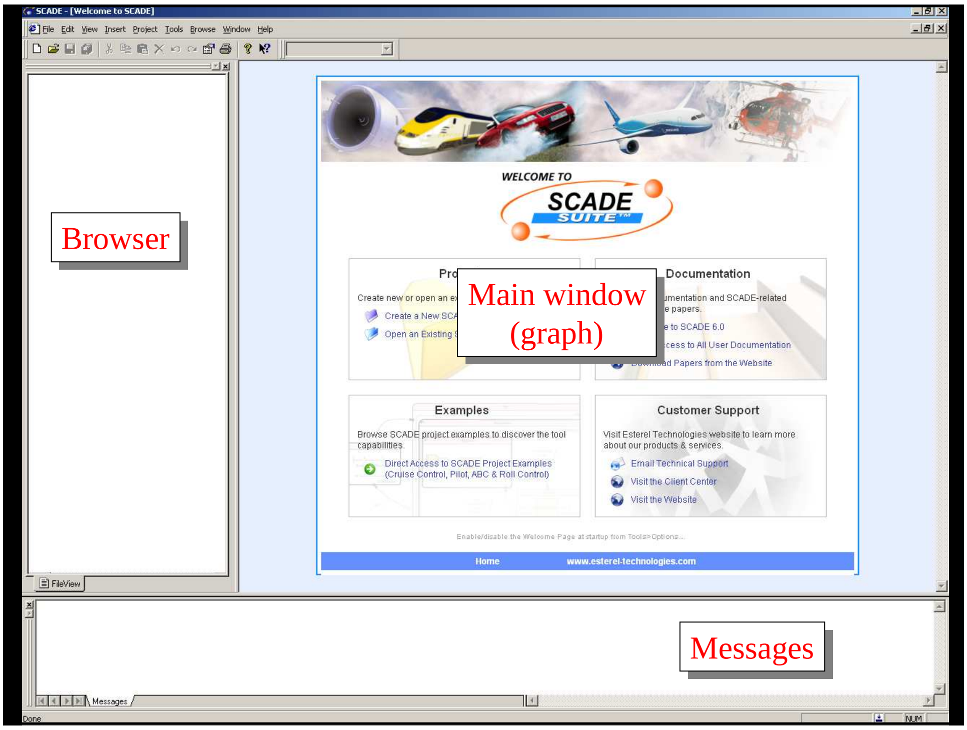
Note type files: $(SCADE)/lib/DefaultAty.aty

Save version: current version

☑ New variable symbols

Overview

- Synchronous model ☞
- Introduction to the Scade language ☞
- Editing a Scade model ☞
- Activation conditions ☞
- Automata ☞
- Arrays ☞
- Iterations ☞
- Global flows: Sensors and probes ☞
- Genericity ☞



Create new or open an ex

Create a New SCA

Open an Existing S

umentation and SCADE-related e papers.

e to SCADE 6.0

ccess to All User Documentation

Download Papers from the Website

Browse SCADE project examples to discover the tool capabilities.

Direct Access to SCADE Project Examples (Cruise Control, Pilot, ABC & Roll Control)

Visit Esterel Technologies website to learn more about our products & services.

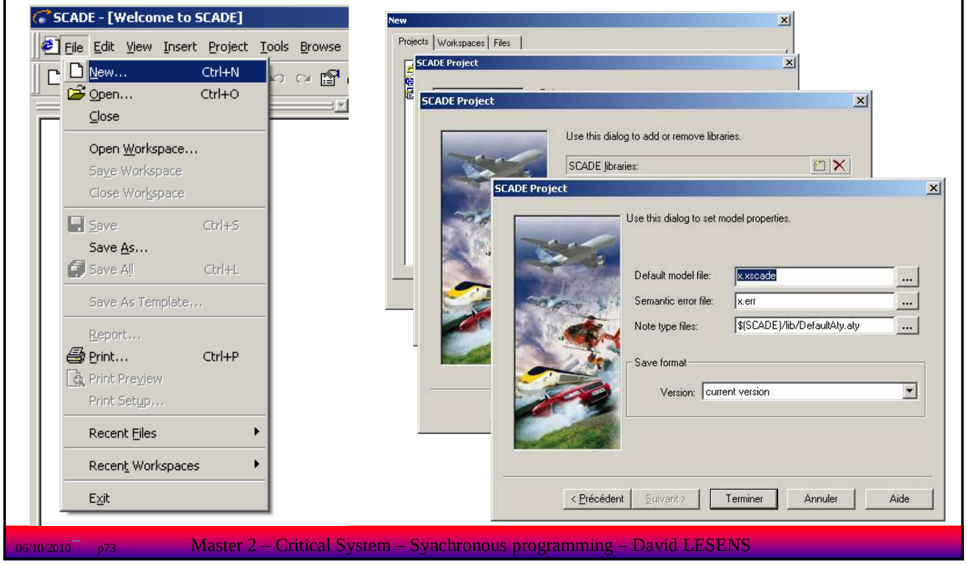
Email Technical Support

Visit the Client Center

Visit the Website

Enable/disable the Welcome Page at startup from Tools>Options...

Creating a new project



Use this dialog to add or remove libraries.

SCADE libraries:

Use this dialog to set model properties.

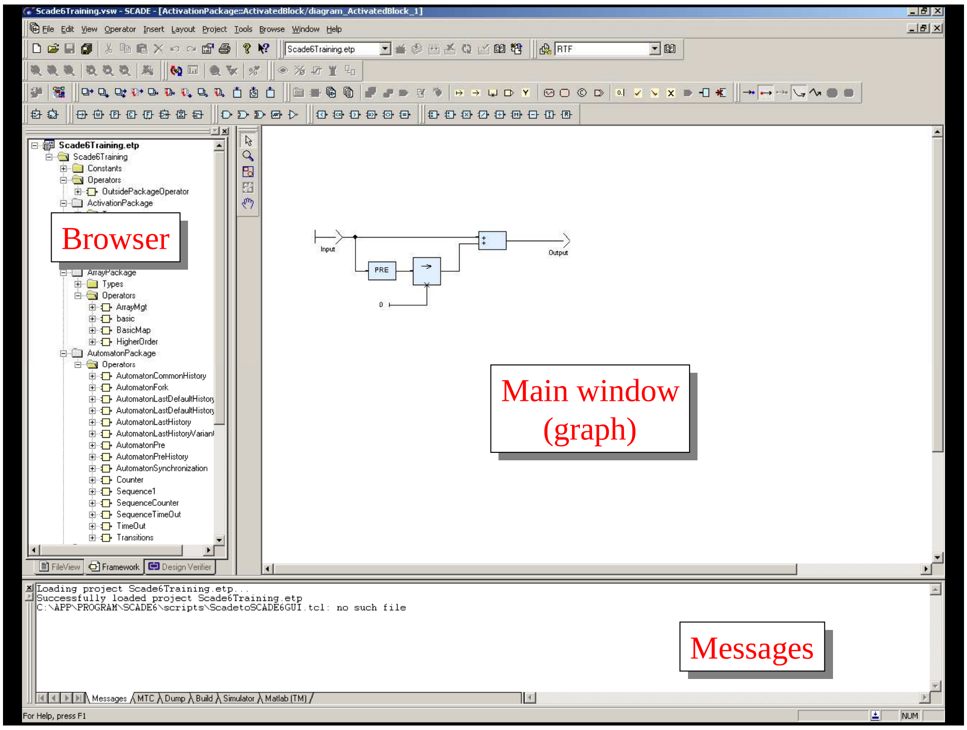
Default model file: x.xscade

Semantic error file: x.err

Note type files: $(SCADE)/lib/DefaultAty.aty

Save format

Version: current version



Browser

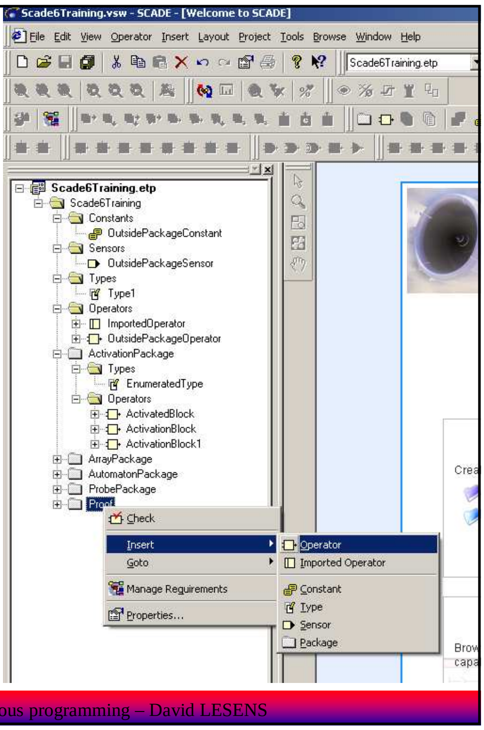
Main window (graph)

Messages

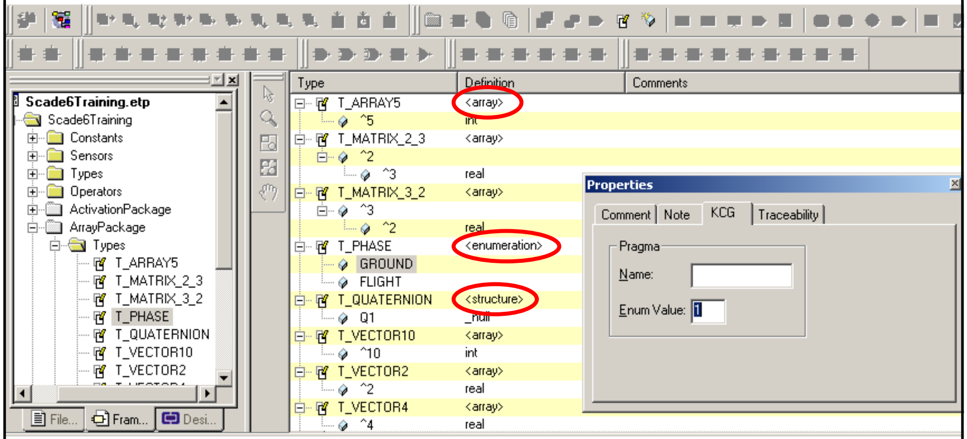
```
Loading project Scade6Training.etp...
Successfully loaded project Scade6Training.etp
C:\APP\PROGRAM\SCADE6\scripts\ScadetoSCADE6GUI.tcl: no such file
```

Packages

- Definitions of
  - Scade operators
  - Imported operators
  - Constants
  - Types
  - Sensors
  - Packages
- Inside or outside a package



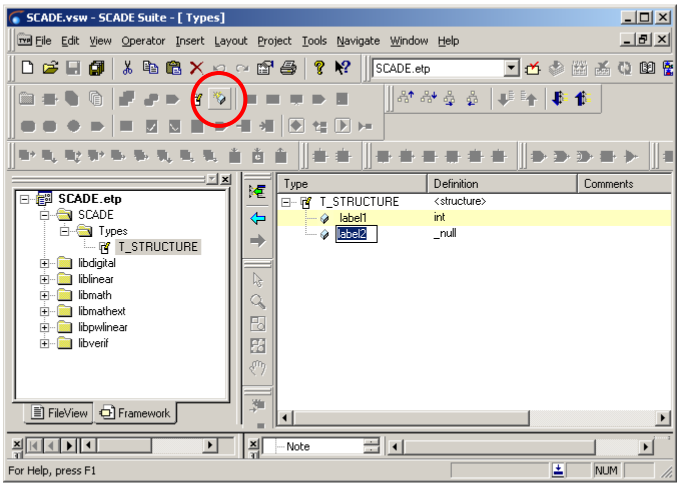
Management of types (1/3)



Type	Definition	Comments
T_ARRAY5	<array>	
^5	int	
T_MATRIX_2_3	<array>	
^2		
^3	real	
T_MATRIX_3_2	<array>	
^3		
^2	real	
T_PHASE	<enumeration>	
GROUND		
FLIGHT		
T_QUATERNION	<structure>	
Q1	_null	
T_VECTOR10	<array>	
^10	int	
T_VECTOR2	<array>	
^2	real	
T_VECTOR4	<array>	
^4	real	

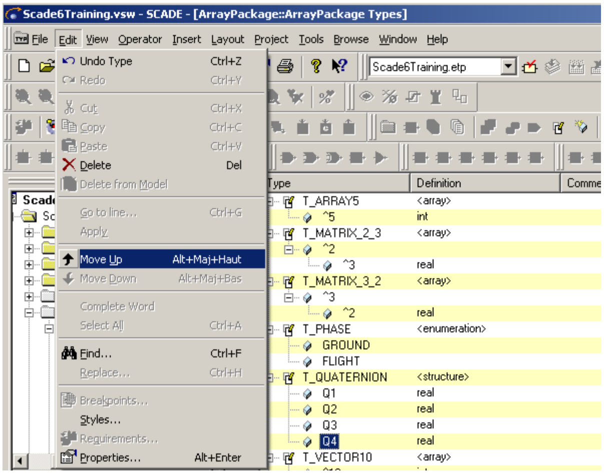
Name:
Enum Value: 1

Management of types (2/3)



Type	Definition	Comments
T_STRUCTURE	<structure>	
label1	int	
label2	_null	

Management of types (3/3)



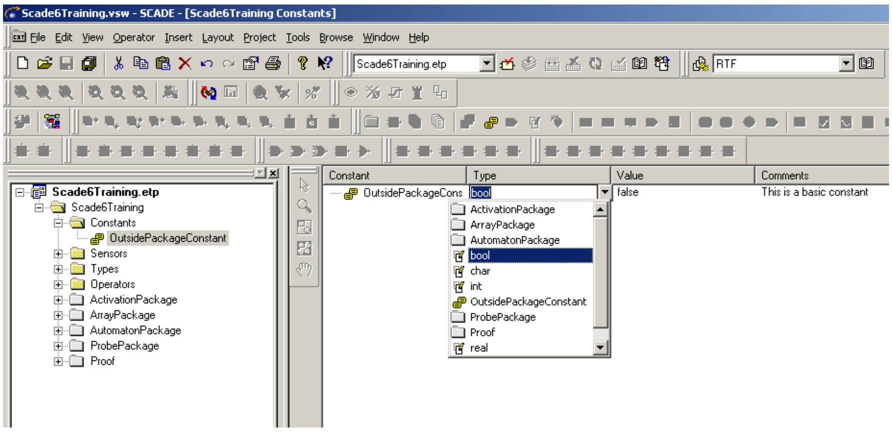
Type	Definition	Comme
T_ARRAY5	<array>	
^5	int	
T_MATRIX_2_3	<array>	
^2		
^3	real	
T_MATRIX_3_2	<array>	
^3		
^2	real	
T_PHASE	<enumeration>	
GROUND		
FLIGHT		
T_QUATERNION	<structure>	
Q1	real	
Q2	real	
Q3	real	
Q4	real	
T_VECTOR10	<array>	

Integers and reals

- Integers
  - Binary 0b01001
  - Octal 0563
  - Decimal 9637
  - Hexadecimal 0xAF6C
- Encoding
  - short, int, long
  - Float, double

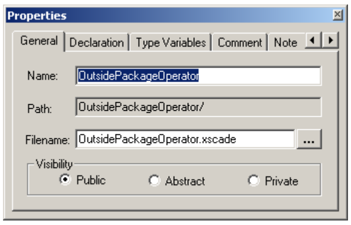
} Shall be defined by the user

Defining a constant



Constant	Type	Value	Comments
OutsidePackageCons	bool	false	This is a basic constant

Changing an object properties

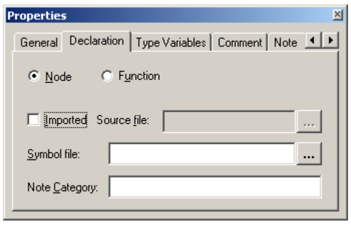


Name: OutsidePackageOperator

Path: OutsidePackageOperator/

Filename: OutsidePackageOperator.xscade

Visibility: ● Public ○ Abstract ○ Private



● Node ○ Function

□ Imported Source file:

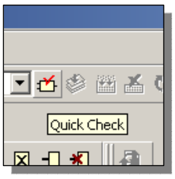
Symbol file:

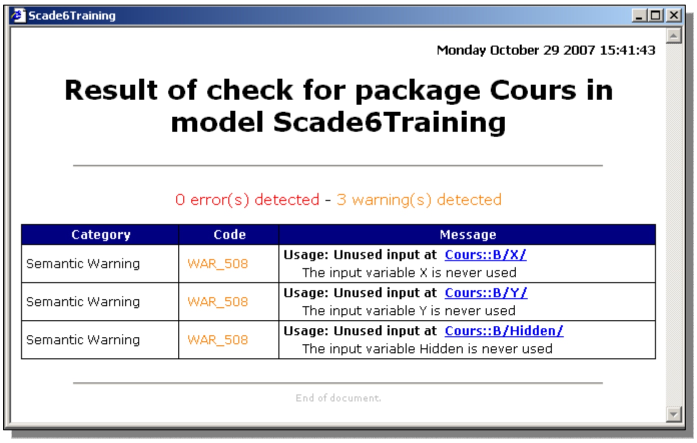
Note Category:

Keyword list

- Scade keywords
  - abstract, activate, and, assume, automaton, bool, case, char, clock, const, default, div, do, else, elsif, emit, end, enum, every, false, fby, final, flatten, fold, foldi, foldw, foldwi, function, guarantee, group, if, imported, initial, int, is, last, let, make, map, mapfold, mapi, mapw, mapwi, match, merge, mod, node, not, numeric, of, onreset, open, or, package, parameter, pre, private, probe, public, real, restart, resume, returns, reverse, sensor, sig, specialize, state, synchro, tel, then, times, transpose, true, type, unless, until, var, when, where, with, xor
- + Targeted programming language keywords

Quick check





Scade6Training

Monday October 29 2007 15:41:43

Result of check for package Cours in model Scade6Training

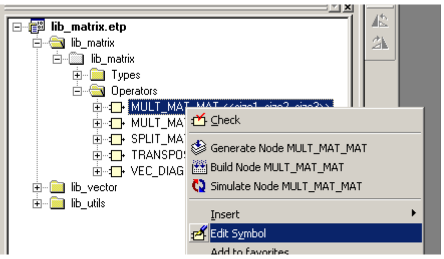
0 error(s) detected - 3 warning(s) detected

Category	Code	Message
Semantic Warning	WAR_508	**Usage: Unused input at** Cours::B/X/ The input variable X is never used
Semantic Warning	WAR_508	**Usage: Unused input at** Cours::B/Y/ The input variable Y is never used
Semantic Warning	WAR_508	**Usage: Unused input at** Cours::B/Hidden/ The input variable Hidden is never used

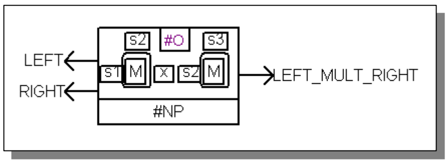
End of document.

The quick check performs syntax and semantics verification
It shall be frequently used

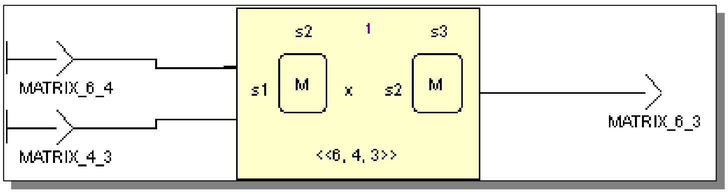
Symbol editor



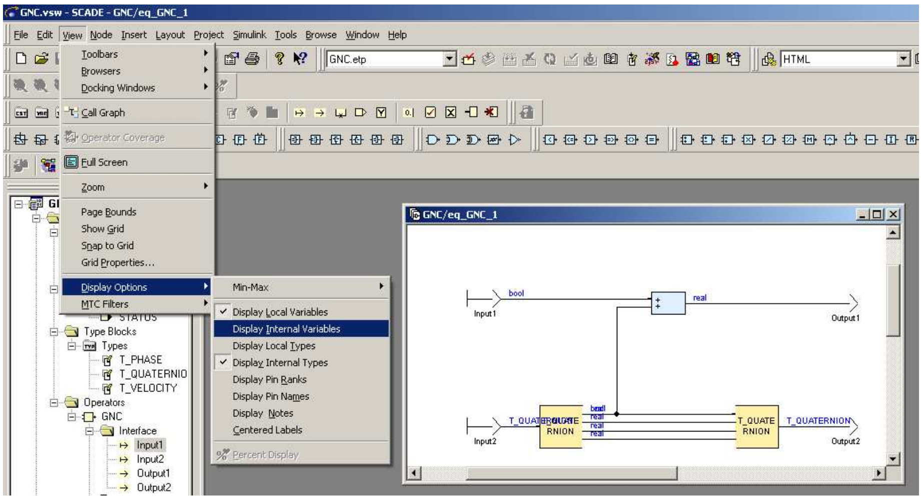
Edition of the symbol



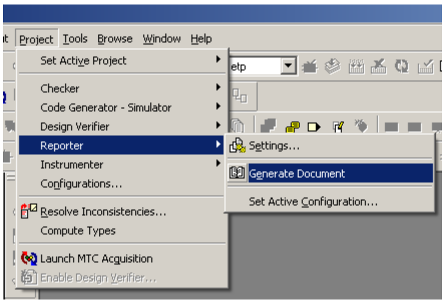
Use of the symbol

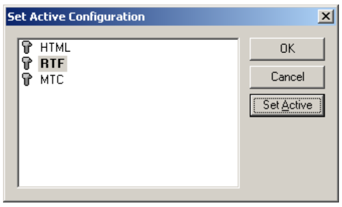


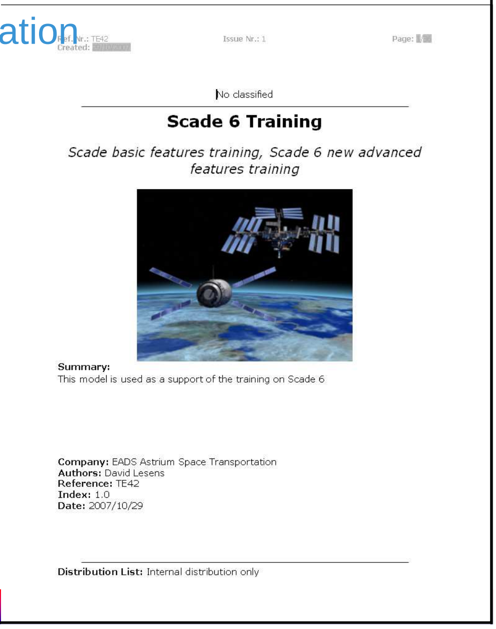
Display types / variable names / …



Generation of documentation






Ref. Nr.: TE42 Issue Nr.: 1 Page: 1/36
Created: 29/10/2007

No classified

Scade 6 Training

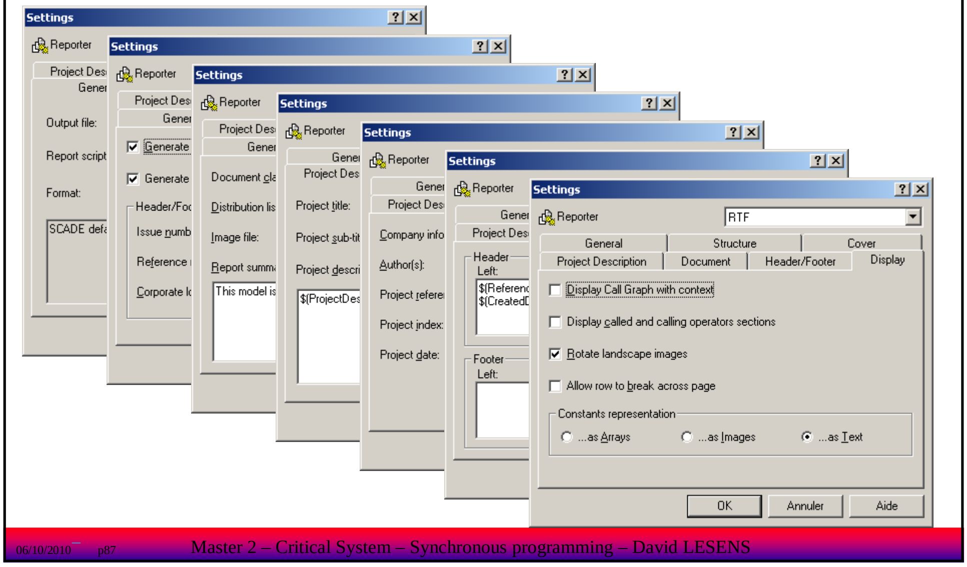
Scade basic features training, Scade 6 new advanced features training

Summary:
This model is used as a support of the training on Scade 6

Company: EADS Astrium Space Transportation
Authors: David Lesens
Reference: TE42
Index: 1.0
Date: 2007/10/29

Distribution List: Internal distribution only

Report customization



Settings

Reporter: RTF

General | Structure | Cover | Project Description | Document | Header/Footer | Display

☐ Display Call Graph with context

☐ Display called and calling operators sections

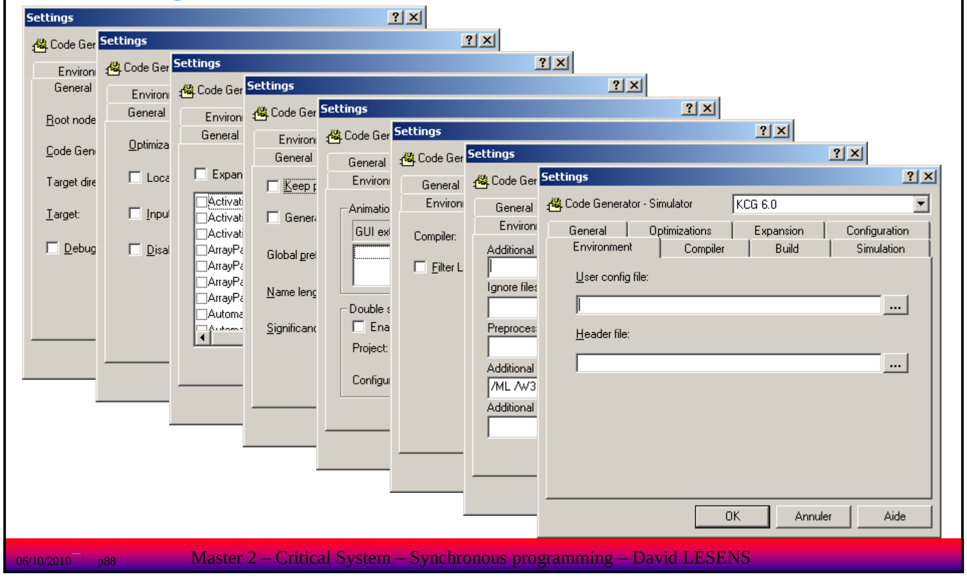
☑ Rotate landscape images

☐ Allow row to break across page

Constants representation

○ ...as Arrays ○ ...as Images ● ...as Text

Code generation customization



Settings

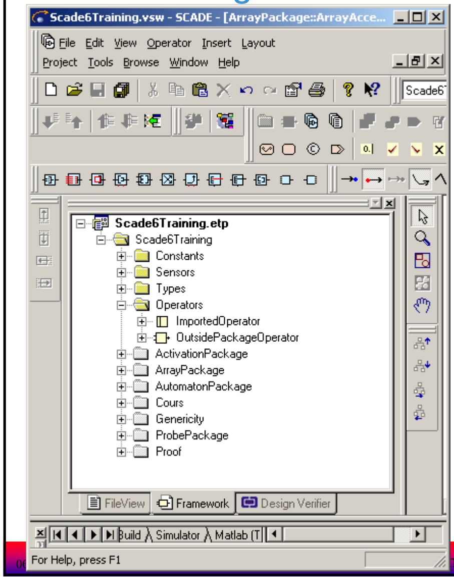
Code Generator - Simulator: KCG 6.0

General | Optimizations | Expansion | Configuration | Environment | Compiler | Build | Simulation

User config file:

Header file:

File management



Scade6Training.xscade
ImportedOperator.xscade
OutsidePackageOperator.xscade
ActivationPackage.xscade
ArrayPackage.xscade
AutomatonPackage.xscade
Cours.xscade
Genericity.xscade
ProbePackage.xscade
Proof.xscade

Documentation

- **Welcome to SCADE 6.o**
- **Getting Started with SCADE**
- **Scade Language Tutorial**
- **Scade Language Primer**
- **Scade Language Reference Manual**
- **SCADE User Manual**
- **SCADE Technical Manual**
- **SCADE Libraries Manual**
- **SCADE UML Metamodel Card**
- **SCADE Gateway for Rhapsody Guidelines**
- **Simulink™ Gateway Guidelines**
- **Simulink™ Modeling Guidelines**
- **RTOS Wrapper Guidelines**

About Requirements Management Gateway documentation, check from RMG interface at Help > Documentation or Coupling Notes

Overview

- Synchronous model ☞
- Introduction to the Scade language ☞
- Editing a Scade model ☞
- **Activation conditions** ☞
- Automata ☞
- Arrays ☞
- Iterations ☞
- Global flows: Sensors and probes ☞
- Genericity ☞

“IF” operator

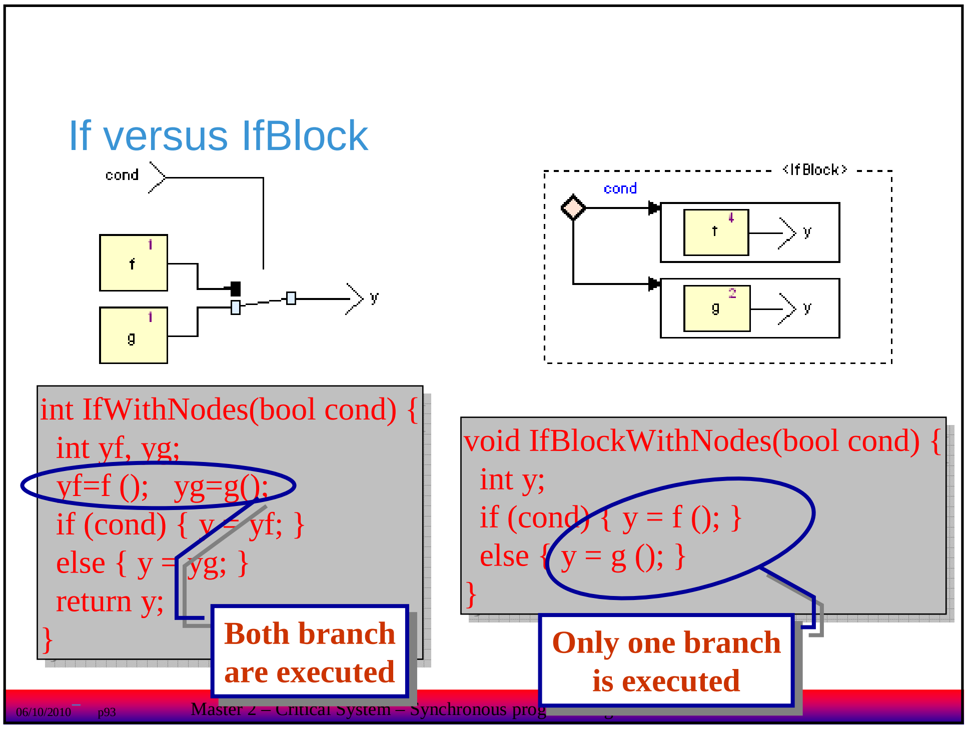
x = if b then y else z

If “b” is true, “x” takes the value “y”,
else, “x” takes the value “z”

Note:

Does not mean

If “b” is true, execute “y”,
else, execute “z”



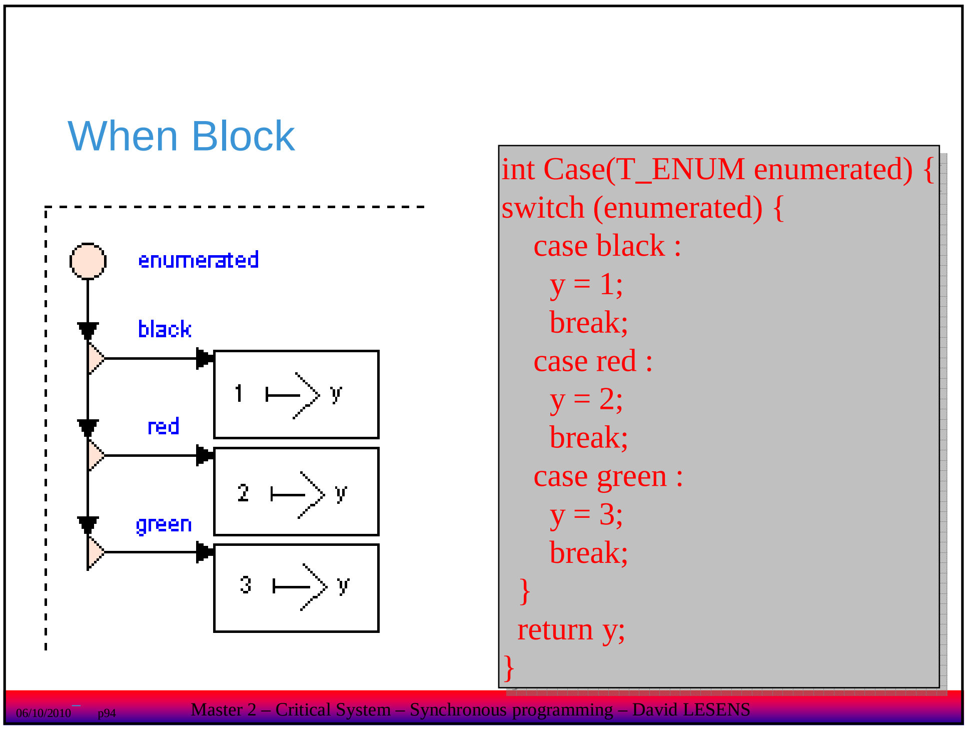
If versus IfBlock

```
int IfWithNodes(bool cond) {
  int yf, yg;
  yf=f ();   yg=g();
  if (cond) { y = yf; }
  else { y = yg; }
  return y;
}
```

Both branch are executed

```
void IfBlockWithNodes(bool cond) {
  int y;
  if (cond) { y = f (); }
  else { y = g (); }
}
```

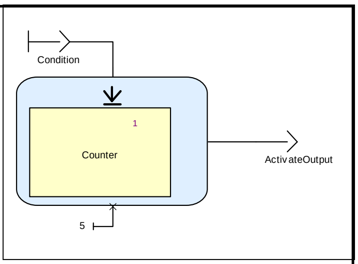
Only one branch is executed



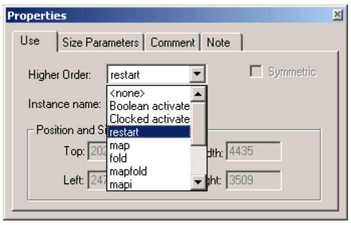
When Block

```
int Case(T_ENUM enumerated) {
switch (enumerated) {
   case black :
     y = 1;
     break;
   case red :
     y = 2;
     break;
   case green :
     y = 3;
     break;
  }
  return y;
}
```

Activation conditions



- Activation condition
  - Condition true = Block activated
  - Condition false = Previous outputs used (was “condact” in Scade 5)
    <u>or</u> Default values
  - Init values before first use
- Restart condition
  - Condition true = Internal memory reset


Properties

Use | Size Parameters | Comment | Note

Higher Order: restart

Instance name:

Position and Size

Top: 20

Left: 24

Width: 4435

Height: 3509

Activation: Example (1/3)

x = a + b, <u>initial</u> default value 5, activation condition c

y = a + b, default value 5, activation condition c

Default values

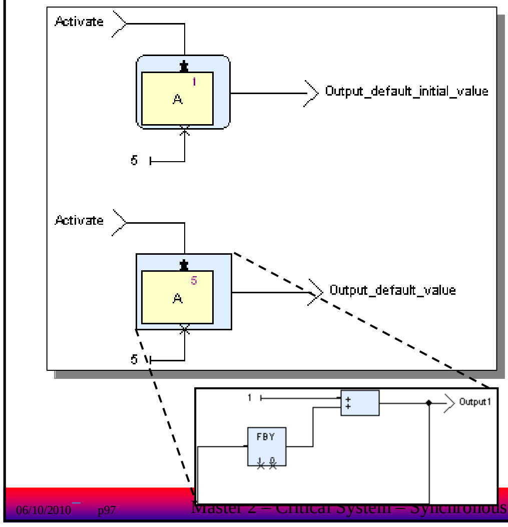
Computed values

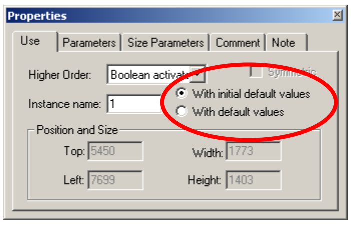
c	0	0	1	0	1
a	3	6	3	2	3
b	3	2	-1	1	4
x	5	5	2	2	7
y	5	5	2	5	7

Last computed values

Default value

Activation: Example (2/3)




Properties

Use | Parameters | Size Parameters | Comment | Note

Higher Order: Boolean activat

□ Symmetric

Instance name: 1

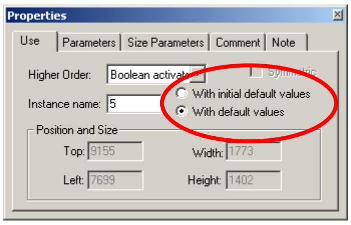
● With initial default values

○ With default values

Position and Size

Top: 5450 Width: 1773

Left: 7699 Height: 1403


Properties

Use | Parameters | Size Parameters | Comment | Note

Higher Order: Boolean activat

□ Symmetric

Instance name: 5

○ With initial default values

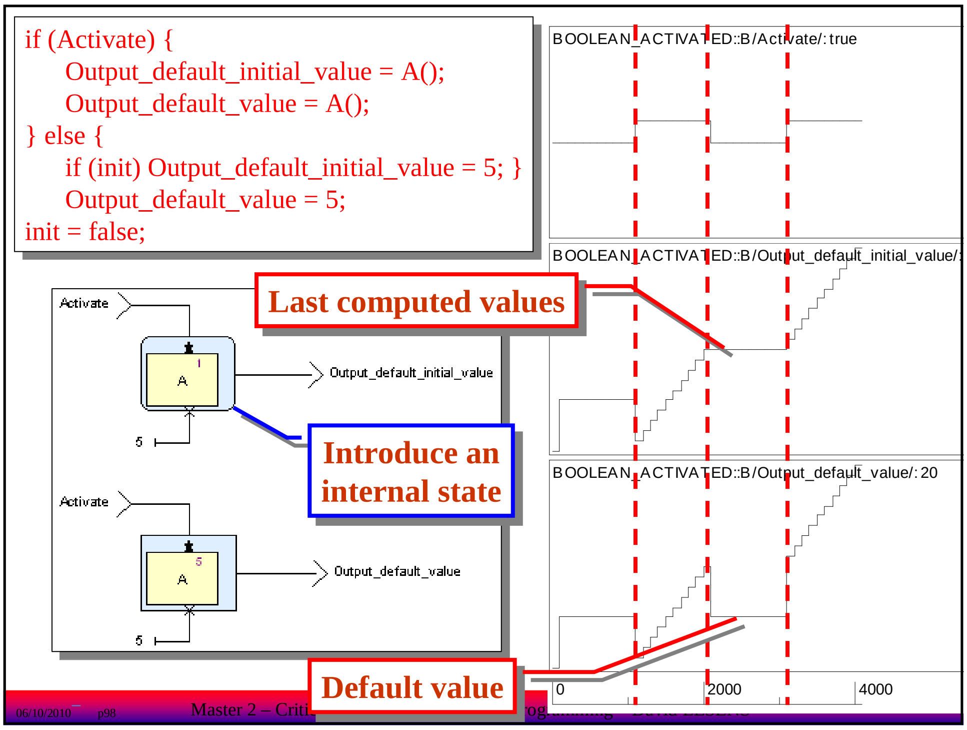
● With default values

Position and Size

Top: 9155 Width: 1773

Left: 7699 Height: 1402

```
if (Activate) {
    Output_default_initial_value = A();
    Output_default_value = A();
} else {
    if (init) Output_default_initial_value = 5; }
    Output_default_value = 5;
init = false;
```

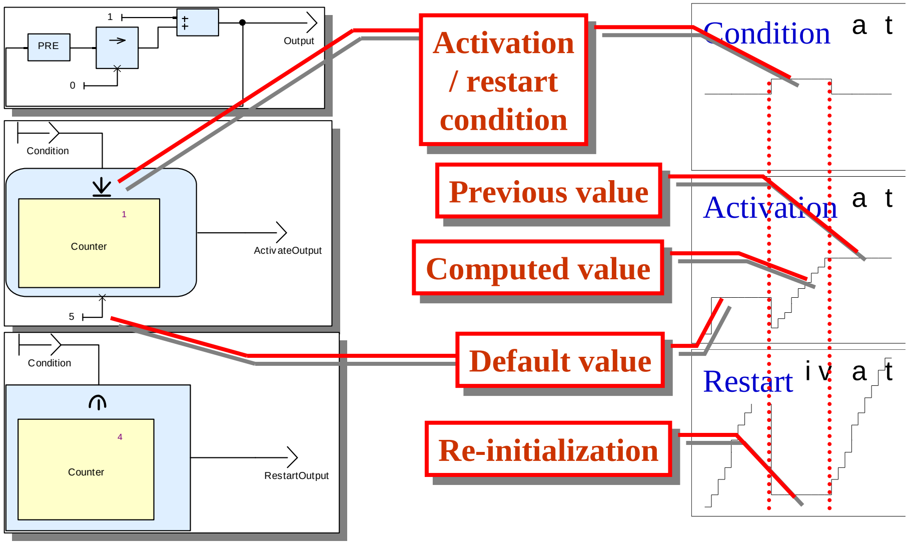


Last computed values

Introduce an internal state

Default value

Activation and restart



Activation / restart condition

Previous value

Computed value

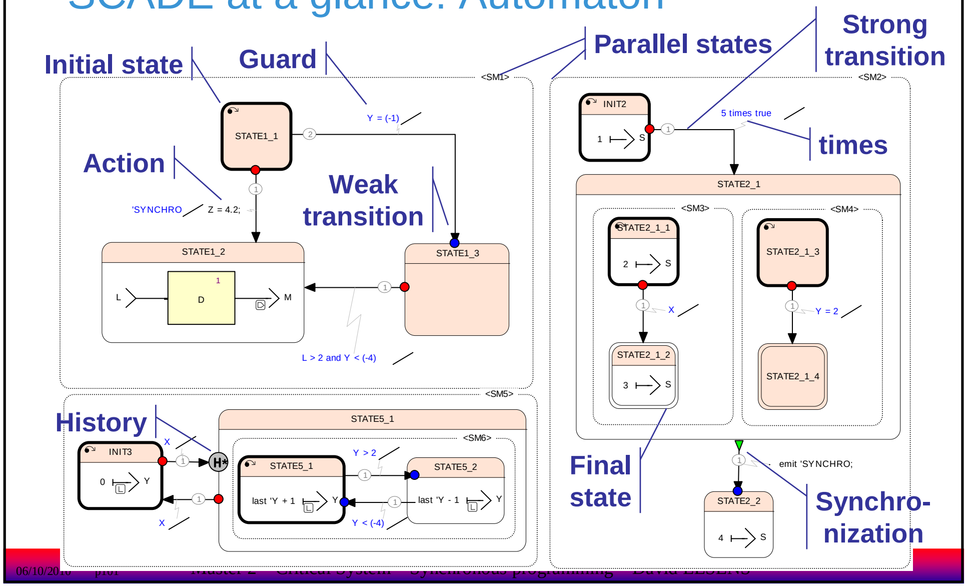
Default value

Re-initialization

Overview

- Synchronous model
- Introduction to the Scade language
- Editing a Scade model
- Activation conditions
- Automata
- Arrays
- Iterations
- Global flows: Sensors and probes
- Genericity

SCADE at a glance: Automaton



Initial state

Guard

Parallel states

Strong transition

Action

Weak transition

times

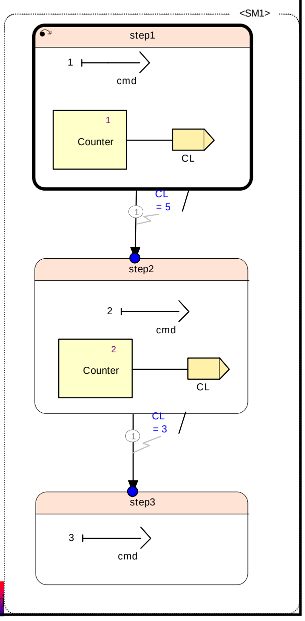
History

Final state

Synchronization

Data flow and automata

- A node is composed of
  - Equations (data-flow)
  - Automata (event driven)
- An automaton is composed of
  - States
  - Transitions
- A state is composed of
  - Equations
  - Automata



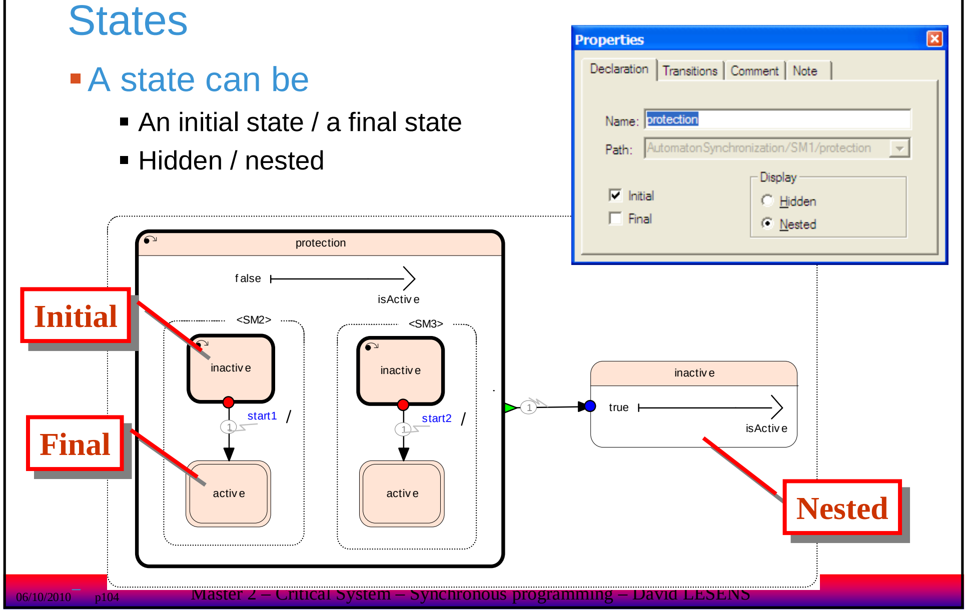
Principles of Automata

- Semantics equivalence
  - There exists a data-flow model semantically equivalent to any automaton
- Automaton scheduling
  - At most one transition fired *per cycle*
  - Exactly one active state per *cycle*

    (except then parallel states are defined)

States

- A state can be
  - An initial state / a final state
  - Hidden / nested



Properties

Declaration | Transitions | Comment | Note

Name: protection

Path: AutomatonSynchronization/SM1/protection

☑ Initial

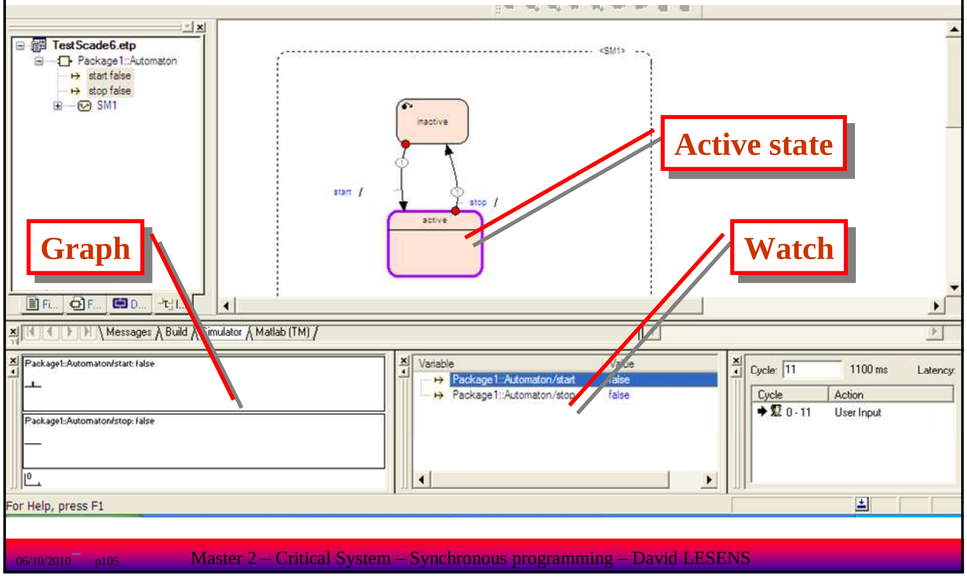
□ Final

Display

○ Hidden

● Nested

Automaton simulation



Active state

Graph

Watch

Package1::Automaton/start: false

Package1::Automaton/stop: false

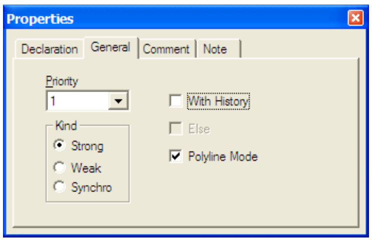
Variable	Value
Package1::Automaton/start	false
Package1::Automaton/stop	false

Cycle: 11 1100 ms Latency:

Cycle	Action
0 - 11	User Input

Transitions

- A transition can
  - have a **weak** pre-emption
  - have a **strong** pre-emption
  - be **synchronized**
- It can have
  - A guard
  - An action
- It has a priority
- It can be with or without a history

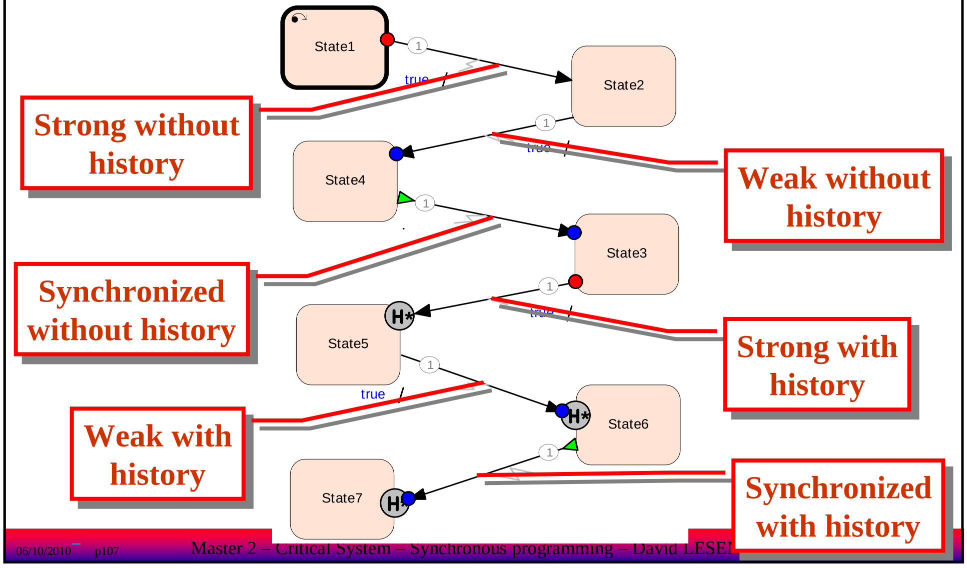


Priority: 1

Kind:
● Strong
○ Weak
○ Synchro

□ With History
□ Else
☑ Polyline Mode

Graphical transitions



Strong without history

Weak without history

Synchronized without history

Strong with history

Weak with history

Synchronized with history

Strong and weak transitions

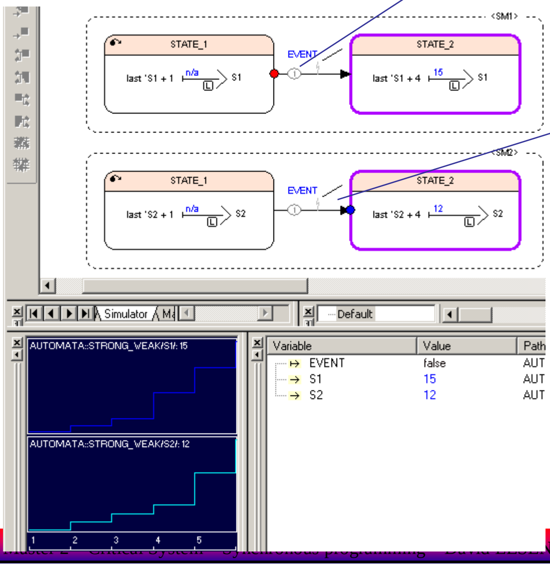
- Strong transition
  - The transition is triggered before the state execution
  - ➔ The guard can not depend on the current value of a data
- Weak transition

  (or “*weak delayed*”)
  - The state is executed before the transition triggering
  - ➔ The guard can depend on the current value of a data

Strong and weak transition

Strong transition

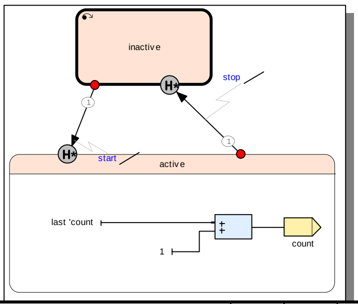
Weak transition



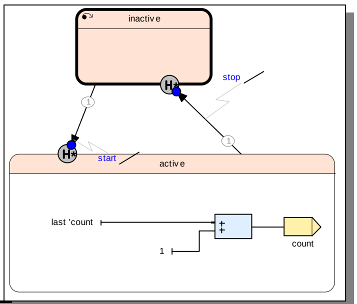
Variable	Value	Path
EVENT	false	AUT
S1	15	AUT
S2	12	AUT

Example (1/2)

Strong transition



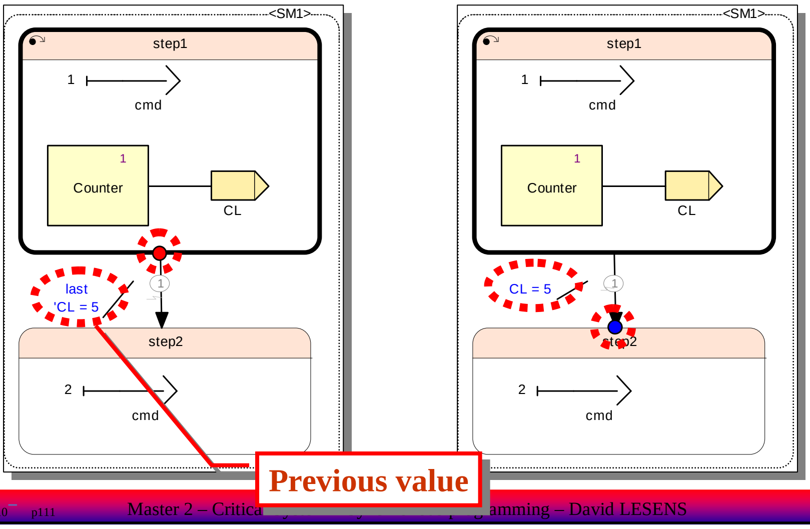
Weak transition



start				T							
stop								T			
count strong	0	0	0	1	2	3	4	4	4	4	4
count weak	0	0	0	0	1	2	3	4	4	4	4

Example (2/2)

The behaviours of the two following models are equivalent



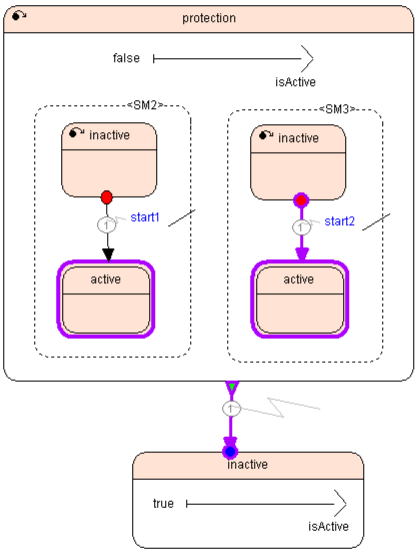
Previous value

Synchronized transition

- A synchronized transition
  - Has no guard
  - Is triggered as soon as all nested automata reach a final state

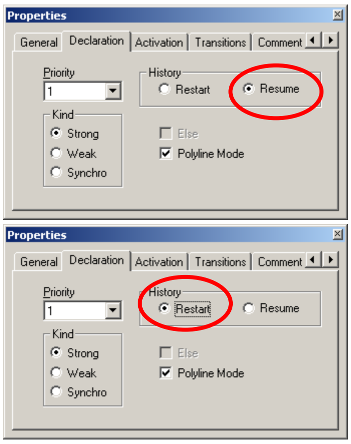
Example

- **Start1 received**
  - Still in protection state
- **Start2 received**
  - Final states reached
- **Transition inactive triggered**



Transition history

- Transition without history
  - The state resumes its execution
  - The memories are reset
- Transition with history
  - The state resumes its execution
  - The memories are not reset
- Two types of memories
  - PRE : local to the state
  - LAST : common to the node


Properties

General | Declaration | Activation | Transitions | Comment

Priority: 1

History: ○ Restart ● Resume

Kind: ● Strong ○ Weak ○ Synchro

□ Else

☑ Polyline Mode

Properties

General | Declaration | Activation | Transitions | Comment

Priority: 1

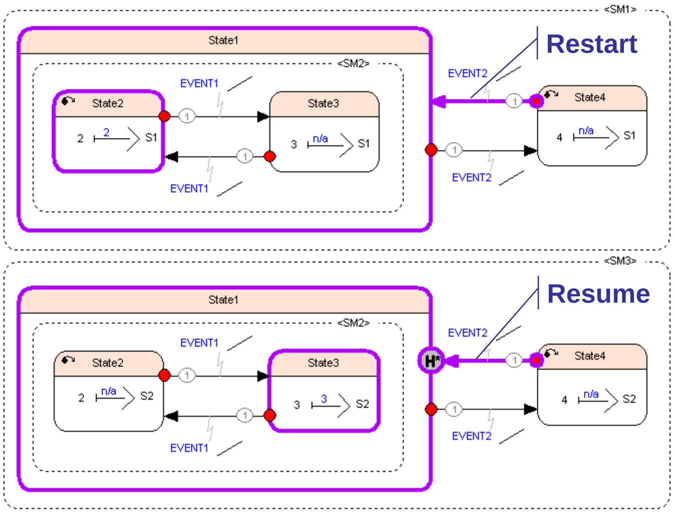
History: ● Restart ○ Resume

Kind: ● Strong ○ Weak ○ Synchro

□ Else

☑ Polyline Mode

Transition with history

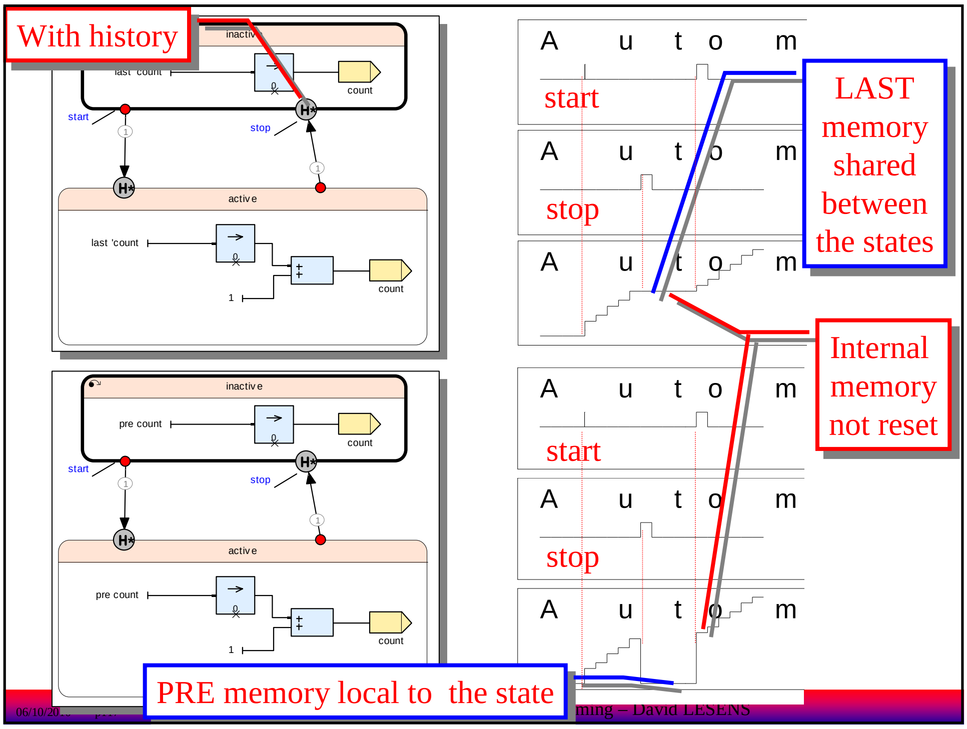


Shared memory

- Data flow point of view
  - Access to the last value of a flow in its scope
    - “pre *expression* ”
- Mode automata point of view
  - Access to values computed in other states
    - “last ‘x”

  (“x” is a named flow, not an expression

  ➔ utilization of ‘)



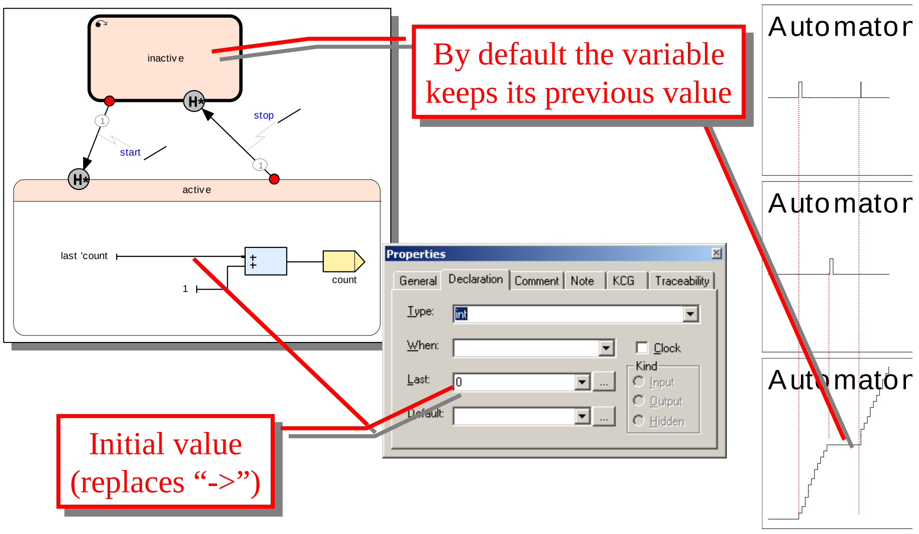
With history

LAST memory shared between the states

Internal memory not reset

PRE memory local to the state

Default actions in state



Type: int

When:

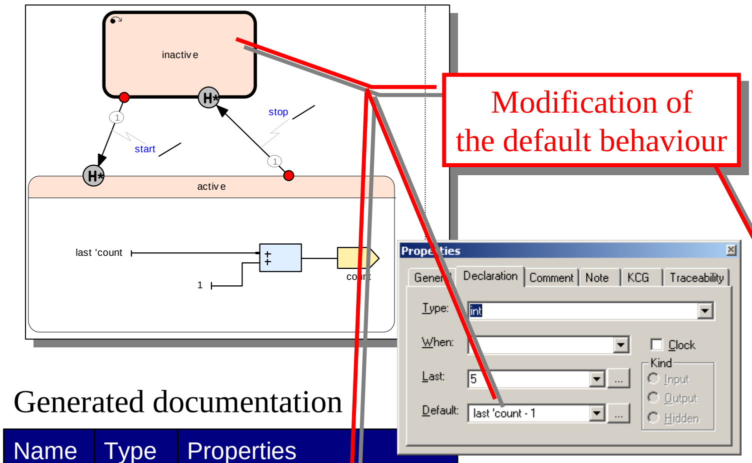
Last: 0

Default:

By default the variable keeps its previous value

Initial value (replaces “->”)

Modifying the default action



Properties

General | Declaration | Comment | Note | KCG | Traceability

Type: int

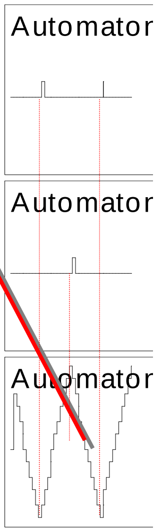
When:

☐ Clock

Kind
○ Input
○ Output
○ Hidden

Last: 5

Default: last 'count - 1

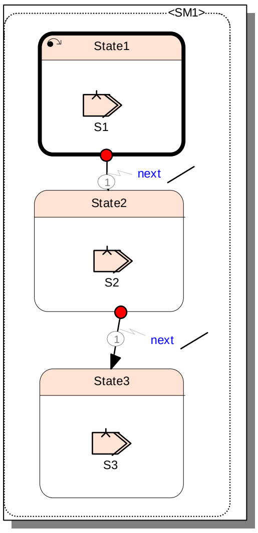


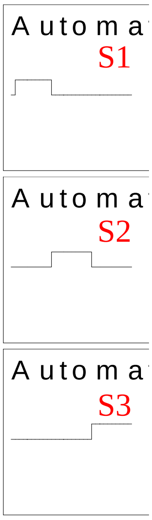
Generated documentation

Name	Type	Properties	
count	int	default	last 'count - 1
		last	5

Signals

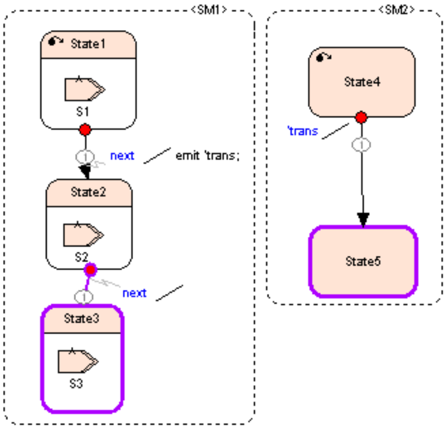
- A signal can be
  - Present ➔ true
  - Absent ➔ false
- A signal can not be
  - An input / output
- ≠ Boolean value
  - A Boolean value keeps its previous value then non updated in a state





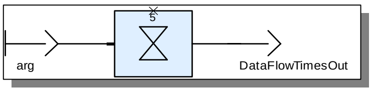
Composition and communication

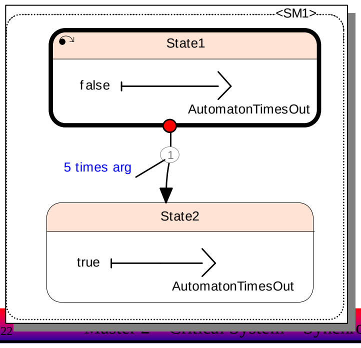
- A signal can be
  - Emitted in a state
  - Emitted on a transition
- A transition can be triggered by a signal

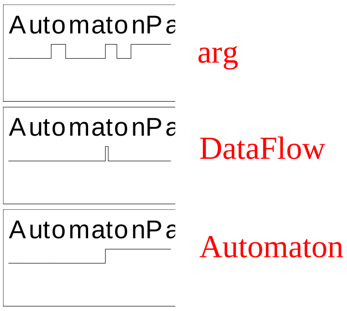


Factor

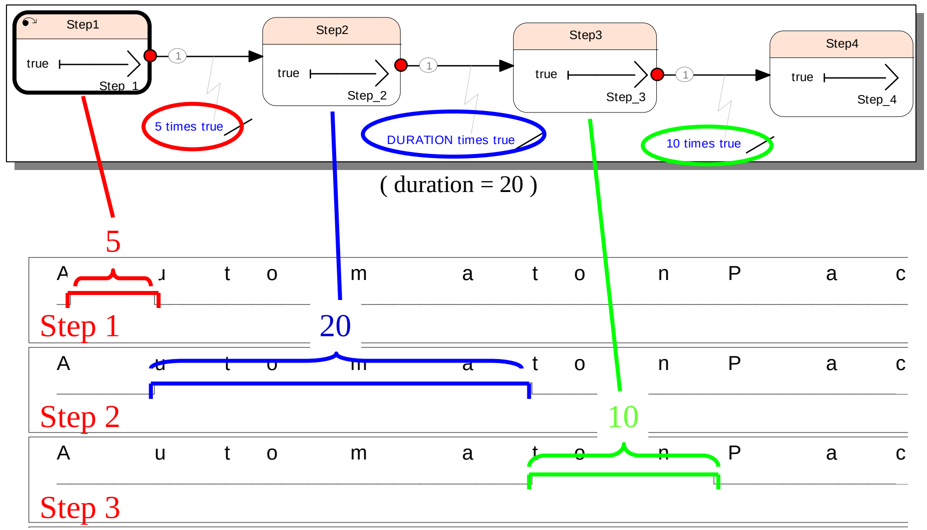
- A factor specifies on many time a condition must be true
  - In a data flow view
  - In a guard (automaton)



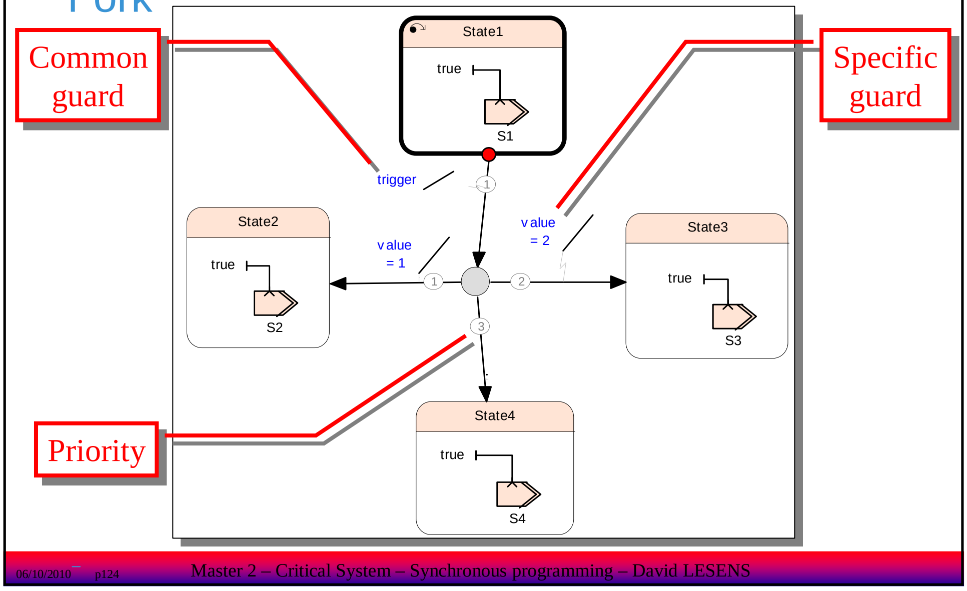




Time-out with factor



Fork

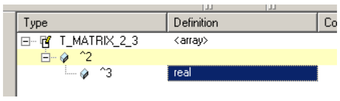


Overview

- Synchronous model ☞
- Introduction to the Scade language ☞
- Editing a Scade model ☞
- Activation conditions ☞
- Automata ☞
- Arrays ☞
- Iterations ☞
- Global flows: Sensors and probes ☞
- Genericity ☞

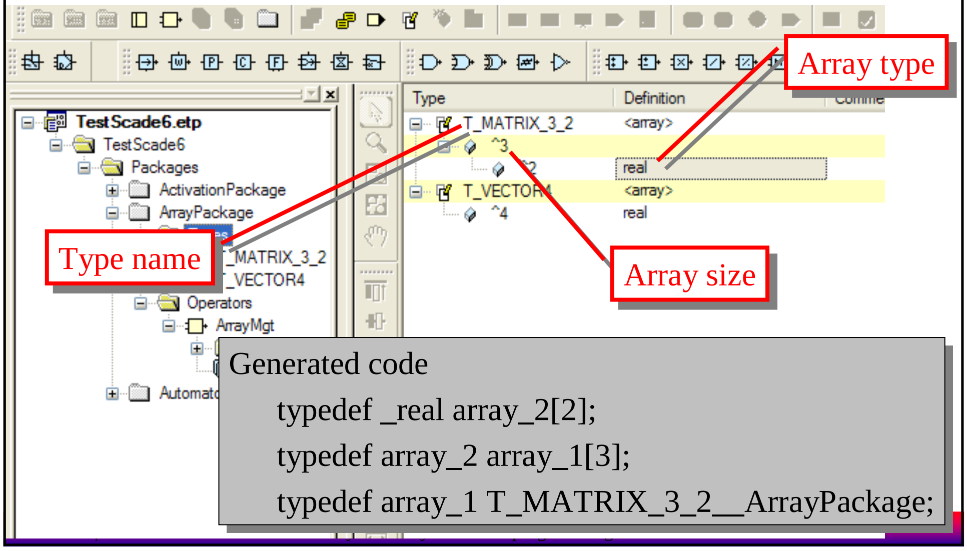
Arrays definition

- Restrictions
  - Static size
  - First element = index 0
- Definitions
  - type VECTOR = real ^ 4 ;
  - type MATRIX_2_3 = real ^ 3 ^ 2 ;
    - 2 lines, 3 columns
    - typedef real LINE_3[3];
    - typedef LINE_3 MATRIX_2_3 [2];



Type	Definition	Co
T_MATRIX_2_3	<array>	
^2		
^3	real	

Editing array types



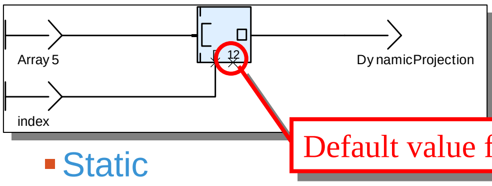
Generated code

```
typedef _real array_2[2];
typedef array_2 array_1[3];
typedef array_1 T_MATRIX_3_2__ArrayPackage;
```

Array access

Array5=[2,4,6,8,10], Index=3

- Dynamic

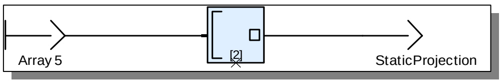


Index = 3 ➔ Output = 8

Index = 10 ➔ Output = 12

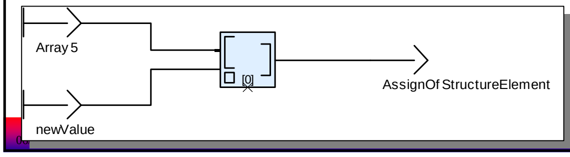
Default value for index out of range

- Static



Output = 6

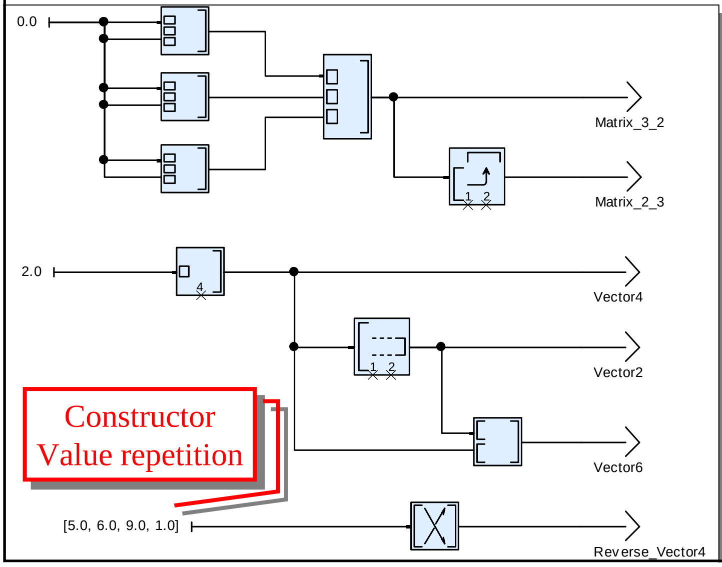
- Assignment



newValue = 3 ➔

Output = [3,4,6,8,10]

Some operators on arrays



Constructor
Value repetition

Data to Vector

Transpose of an Array

Scalar to Vector

Slice of a vector

Concatenation of Arrays

Reverse of a Vector

Overview

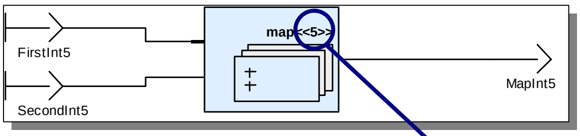
- Synchronous model
- Introduction to the Scade language
- Editing a Scade model
- Activation conditions
- Automata
- Arrays
- Iterations
- Global flows: Sensors and probes
- Genericity

Iterations

- Equivalent to “for” in C

Map / Mapi / Mapw / Mapiw
Fold / Foldi / Foldi / Foldiw

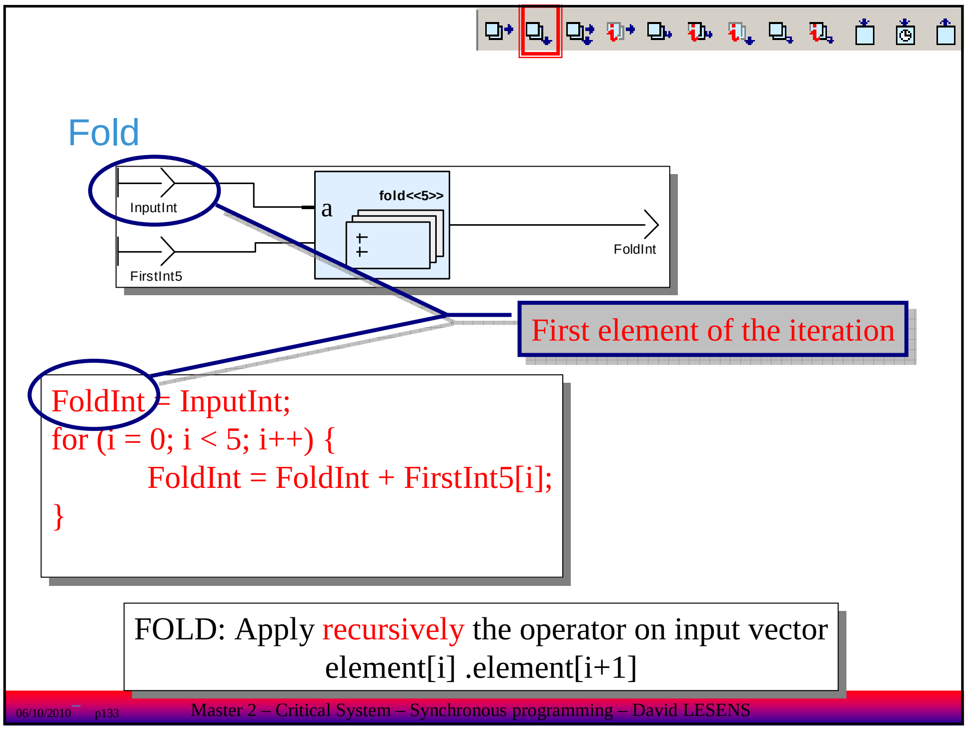
Map



Size of the input vector

```
for (i = 0; i < 5; i++) {
        MapInt5[i] = FirstInt5[i] + SecondInt5[i];
}
```

MAP: Apply the operator successively on each element of the input vector(s) element[i] .element’[i]

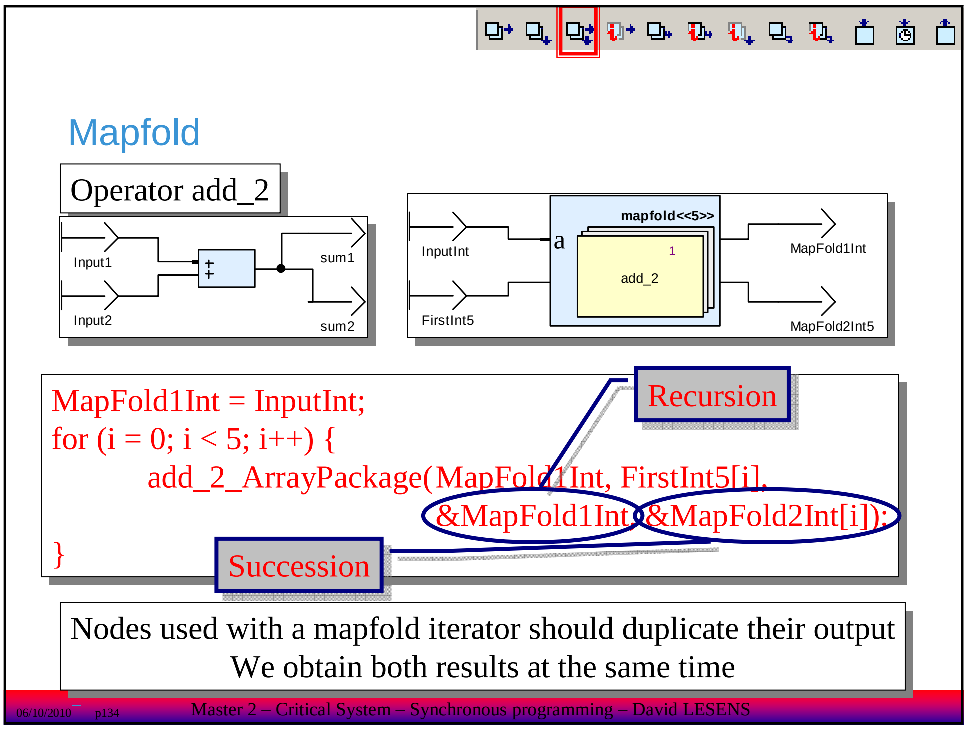


Fold

First element of the iteration

```
FoldInt = InputInt;
for (i = 0; i < 5; i++) {
        FoldInt = FoldInt + FirstInt5[i];
}
```

FOLD: Apply recursively the operator on input vector element[i] .element[i+1]



Mapfold

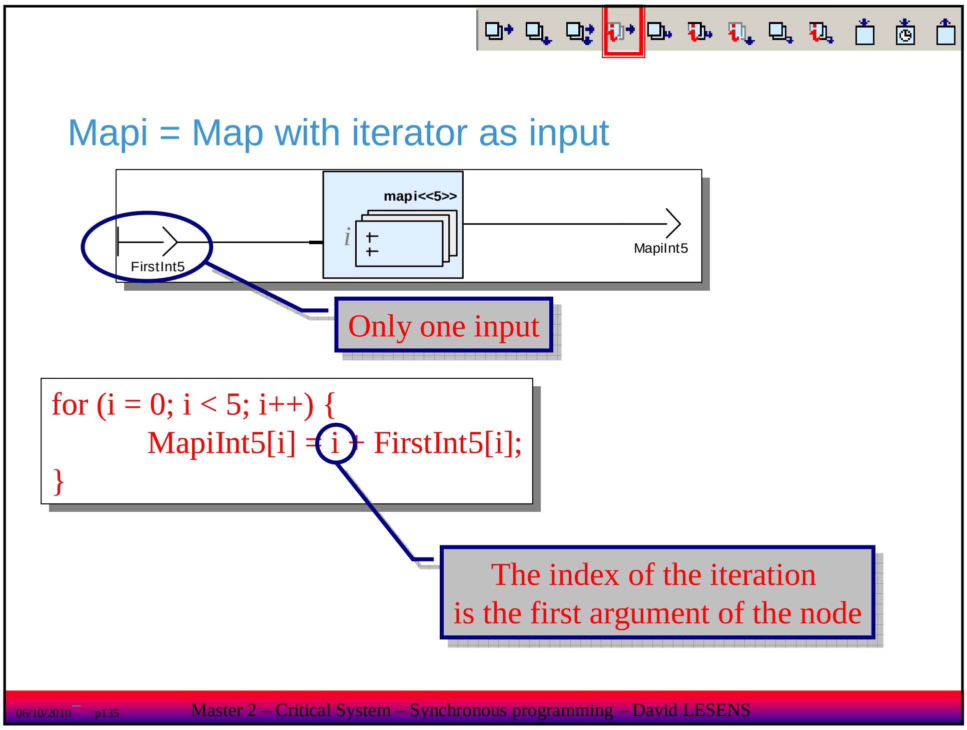
Operator add_2

```
MapFold1Int = InputInt;
for (i = 0; i < 5; i++) {
        add_2_ArrayPackage(MapFold1Int, FirstInt5[i],
                &MapFold1Int,&MapFold2Int[i]);
}
```

Recursion

Succession

Nodes used with a mapfold iterator should duplicate their output
We obtain both results at the same time

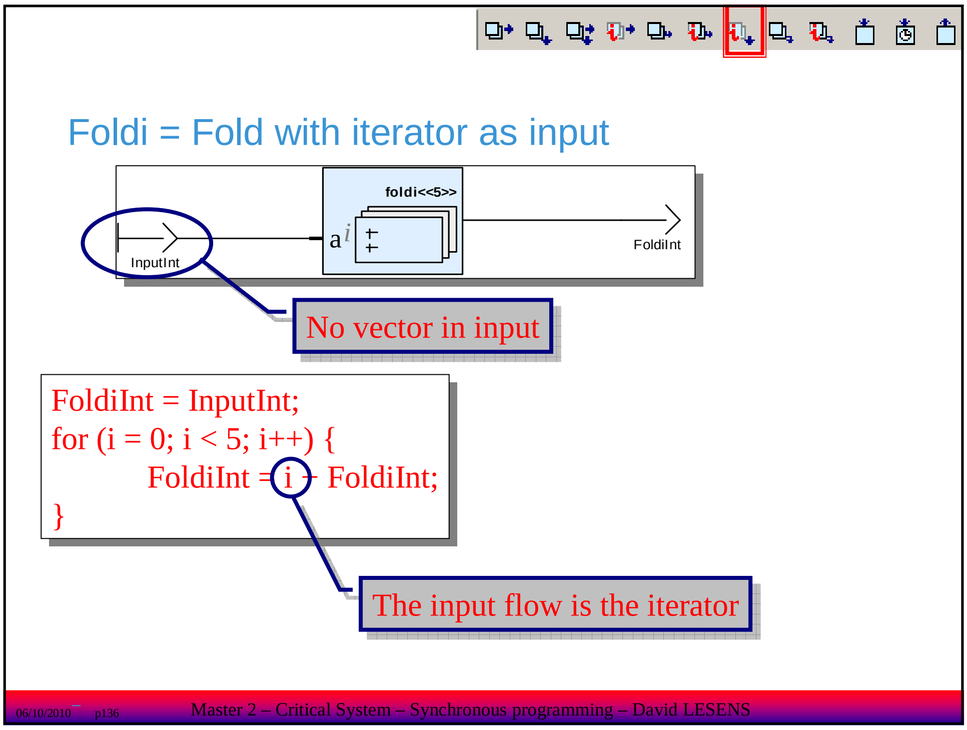


Mapi = Map with iterator as input

Only one input

```
for (i = 0; i < 5; i++) {
        MapiInt5[i] = i + FirstInt5[i];
}
```

The index of the iteration
is the first argument of the node



Foldi = Fold with iterator as input

No vector in input

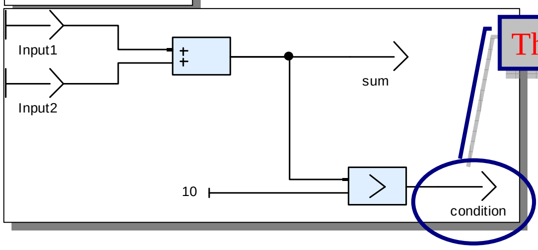
```
FoldiInt = InputInt;
for (i = 0; i < 5; i++) {
        FoldiInt = i + FoldiInt;
}
```

The input flow is the iterator

Mapw / Foldw = Partial operators

- Capability to stop an iteration on a Boolean condition computed by the operator

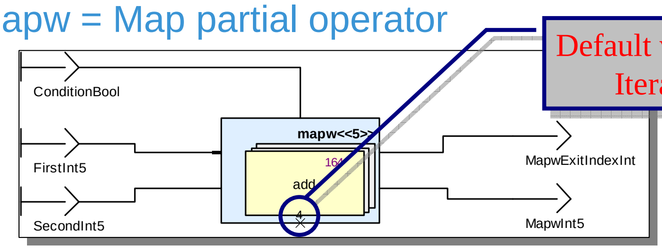
Operator add



The iteration can be stopped

As soon as the condition is false, the iteration is topped

Mapw = Map partial operator



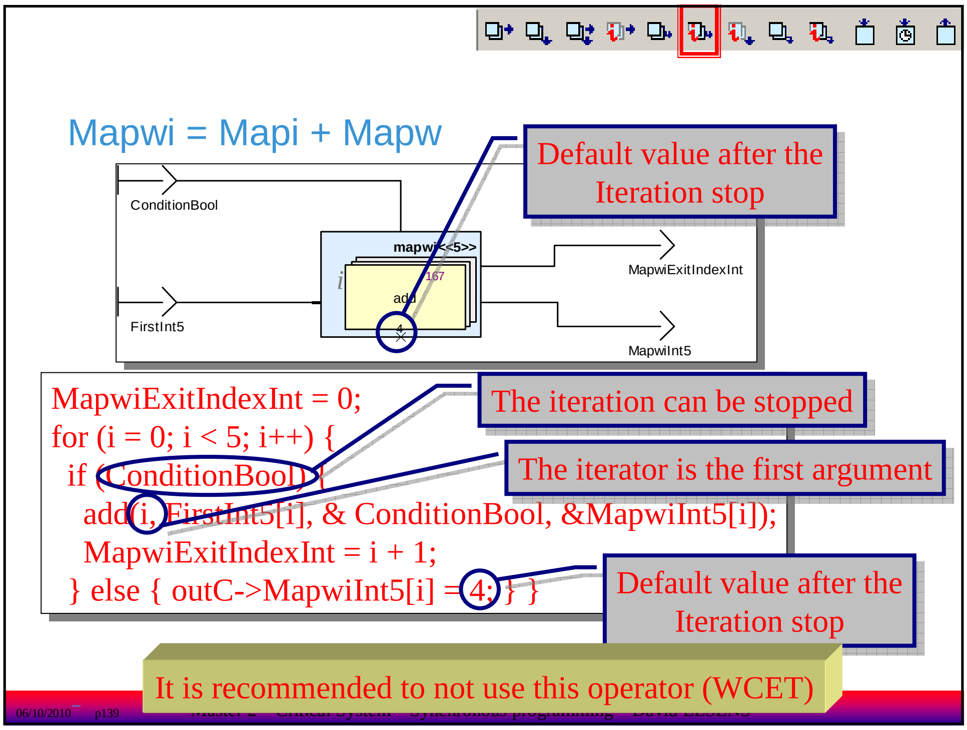
Default value after the Iteration stop

```
MapwExitIndexInt = 0;
for (i = 0; i < 5; i++) {
  if (ConditionBool) {
    add(FirstInt5[i], SecondInt5[i], &ConditionBool, &MapwInt5[i]);
    MapwExitIndexInt = i + 1;
  } else { MapwInt5[i] = 4; } }
```

The iteration can be stopped

Default value after the Iteration stop

It is recommended to not use this operator (WCET)



Mapwi = Mapi + Mapw

Default value after the Iteration stop

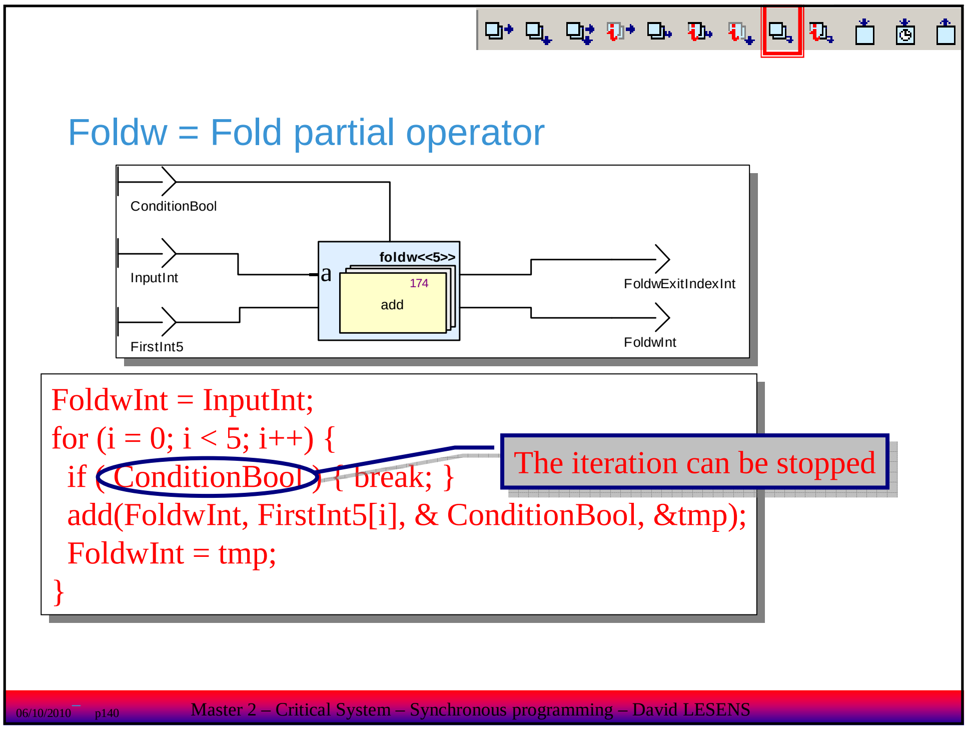
```
MapwiExitIndexInt = 0;
for (i = 0; i < 5; i++) {
  if (ConditionBool) {
    add(i, FirstInt5[i], & ConditionBool, &MapwiInt5[i]);
    MapwiExitIndexInt = i + 1;
  } else { outC->MapwiInt5[i] = 4; } }
```

The iteration can be stopped

The iterator is the first argument

Default value after the Iteration stop

It is recommended to not use this operator (WCET)

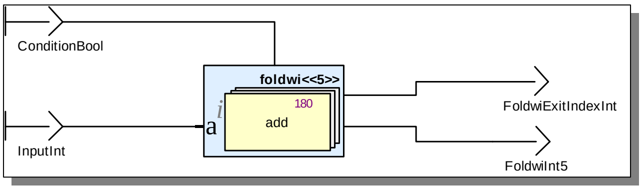


Foldw = Fold partial operator

```
FoldwInt = InputInt;
for (i = 0; i < 5; i++) {
  if (ConditionBool ) { break; }
  add(FoldwInt, FirstInt5[i], & ConditionBool, &tmp);
  FoldwInt = tmp;
}
```

The iteration can be stopped

Foldwi = Foldi + Foldw



```
FoldwiInt5 = InputInt; tmp = ConditionBool;
for (i = 0; i < 5; i++) {
  if ( ConditionBool ) { break; }
  add(i, FoldwiInt5, & ConditionBool, &tmp);
  FoldwiInt5 = tmp;
}
FoldwiExitIndexInt = i;
```

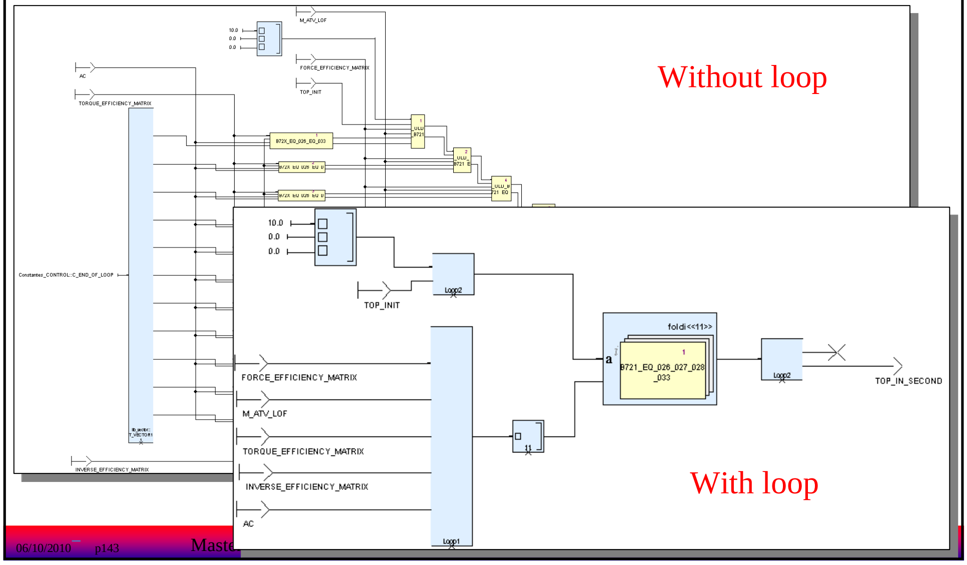
The iteration can be stopped

The input flow is the iterator

Iteration summary

- Map = Successive application
- Fold = Recursive application
- Mapfold = Map + Fold
- Mapi = Map with iterator as input
- Foldi = Fold with iterator as input
- Mapw = Map partial operator
- Mapwi = Mapi + Mapw
- Foldw = Fold partial operator
- Foldwi = Foldi + Foldw

Example 1

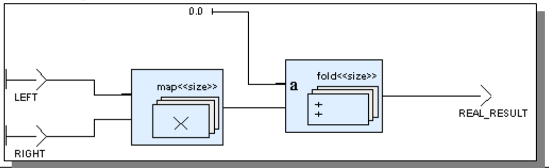


Without loop

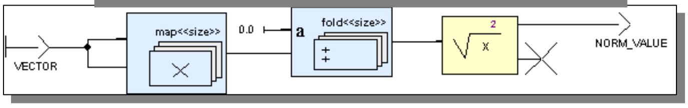
With loop

Example 2: cross product

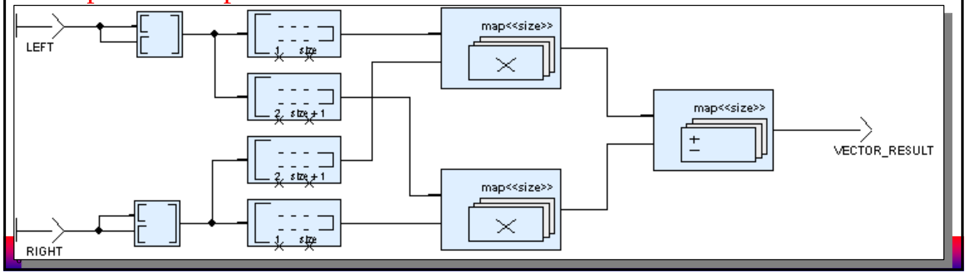
Compute scalar product



Compute vector norm



Compute cross product



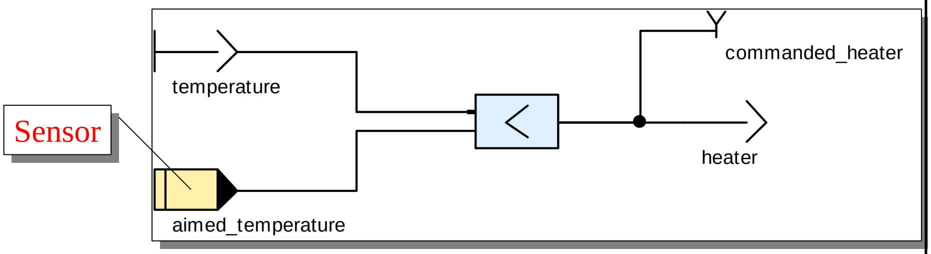
Overview

- Synchronous model
- Introduction to the Scade language
- Editing a Scade model
- Activation conditions
- Automata
- Arrays
- Iterations
- Global flows: Sensors and probes
- Genericity

Sensors

- Sensor: Global system input

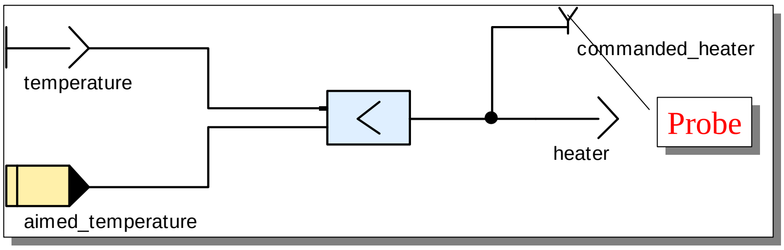
Input temperature
Output heater



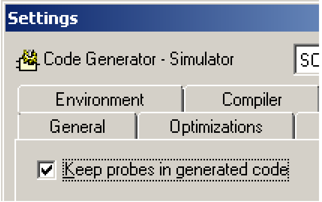
extern _int aimed_temperature__ProbePackage;

Probes

- Probe: Global system output



```
typedef struct {/* context */
_bool heater; /* outputs */
_bool commanded_heater; /* probes */
} C_controller__ProbePackage;
```



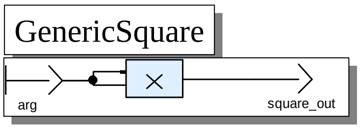
☑ Keep probes in generated code

Overview

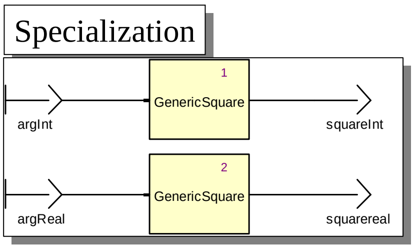
- Synchronous model ☞
- Introduction to the Scade language ☞
- Editing a Scade model ☞
- Activation conditions ☞
- Automata ☞
- Arrays ☞
- Iterations ☞
- Global flows: Sensors and probes ☞
- Genericity ☞

Generic operator definition

GenericSquare



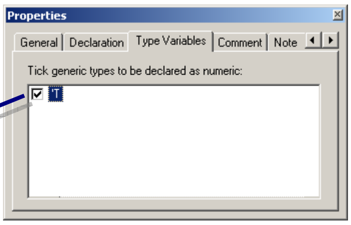
Specialization



	Name	Type
Input	**arg**	**'T**
Output	**square_out**	**'T**

Name	Generic Type
'T	**numeric**

Definition of a generic numeric type



Tick generic types to be declared as numeric:

☑ 'T

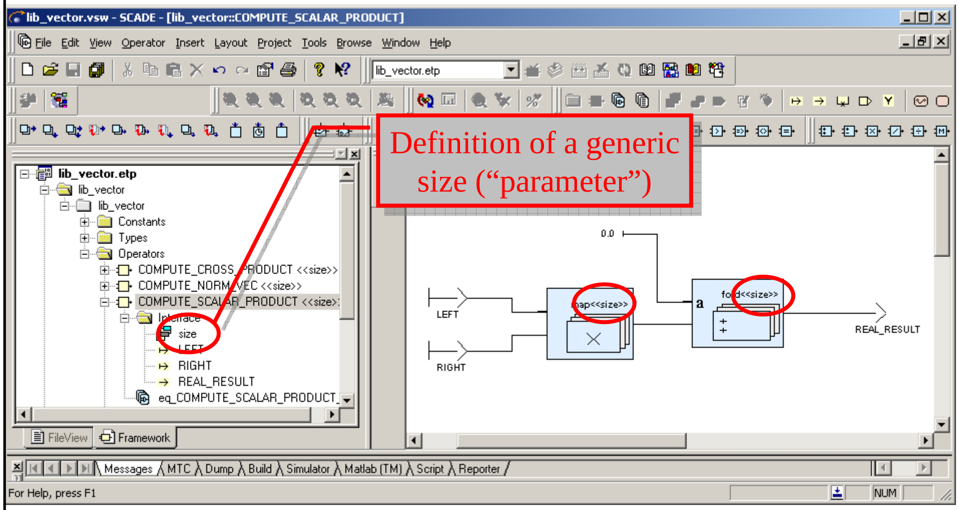
Generic operator instantiation

```
int GenericSquare_int ( int arg ) {
          int square_out;
          square_out = arg * arg;
          return square_out;
}
```

```
real GenericSquare_real ( real arg ) {
          real square_out;
          square_out = arg * arg;
          return square_out;
}
```

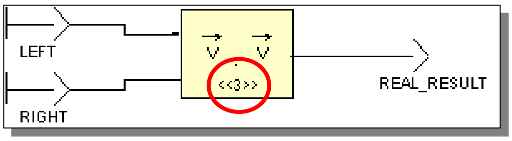
```
void Specialization(    int argInt; real argReal;
                        int squareInt; real squarereal; ) {
*squareReal = GenericSquare_real ( argReal );
*squareInt =  GenericSquare_int ( argInt );
}
```

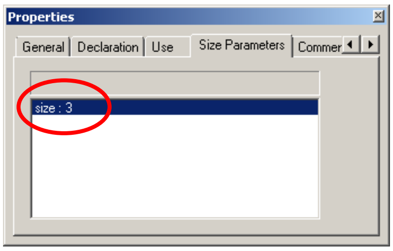
Definition of parameters


lib_vector.vsw - SCADE - [lib_vector::COMPUTE_SCALAR_PRODUCT]
File Edit View Operator Insert Layout Project Tools Browse Window Help
lib_vector.etp
lib_vector
lib_vector
Constants
Types
Operators
COMPUTE_CROSS_PRODUCT <<size>>
COMPUTE_NORM_VEC <<size>>
COMPUTE_SCALAR_PRODUCT <<size>>
Interface
size
LEFT
RIGHT
REAL_RESULT
eq_COMPUTE_SCALAR_PRODUCT_
FileView
Framework
0.0
LEFT
RIGHT
map<<size>>
fold<<size>>
a
REAL_RESULT
Messages MTC Dump Build Simulator Matlab (TM) Script Reporter
For Help, press F1
NUM

Definition of a generic size (“parameter”)

Parameter instantiation





size : 3

```
REAL_RESULT = 0.0;
for (i = 0; i < 3; i++) {
   REAL_RESULT = REAL_RESULT + (*LEFT)[i] * (*RIGHT)[i];
}
return REAL_RESULT;
```

Overview

- Critical real-time embedded software ☞
- Principles of the approach ☞
  - Introduction ☞
  - Formal semantics ☞
- SCADE ☞
- Model validation ☞

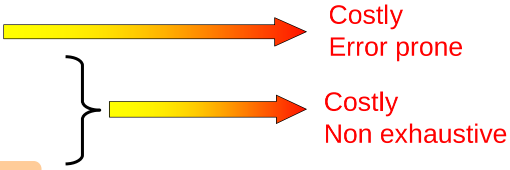
Software validation

Correct software

- No runtime errors
- Satisfaction of real time constraints
- Compliance with the expected results

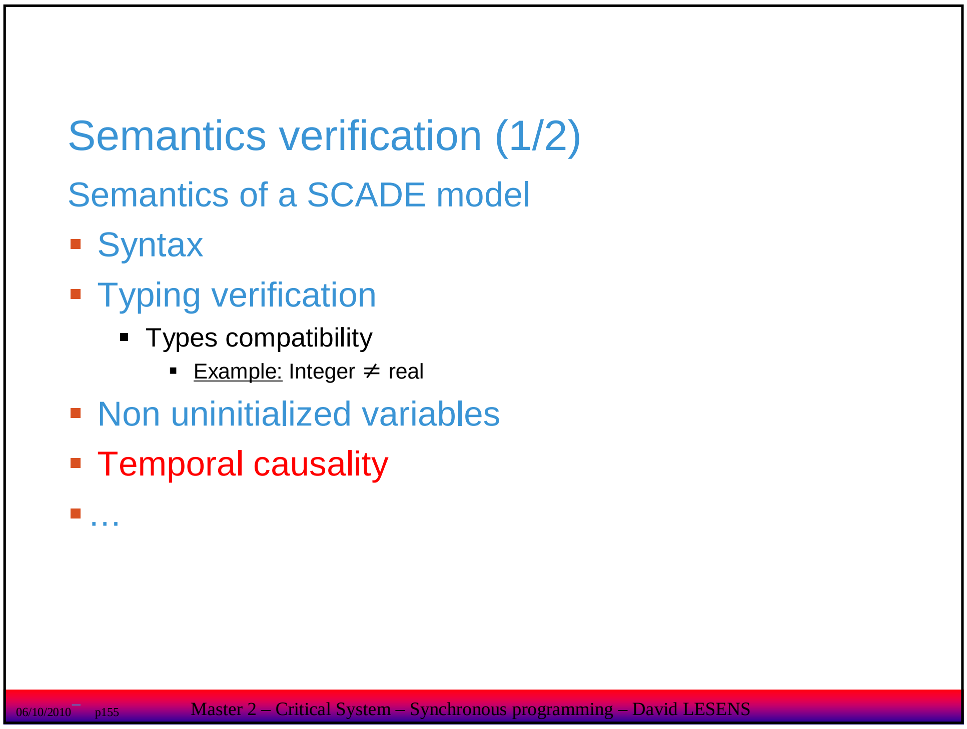
Solutions

- Manual review
- Dynamic testing
  - A code level
  - At model level
- Semantics checking
- Abstract interpretation
- Formal proof



Costly
Error prone

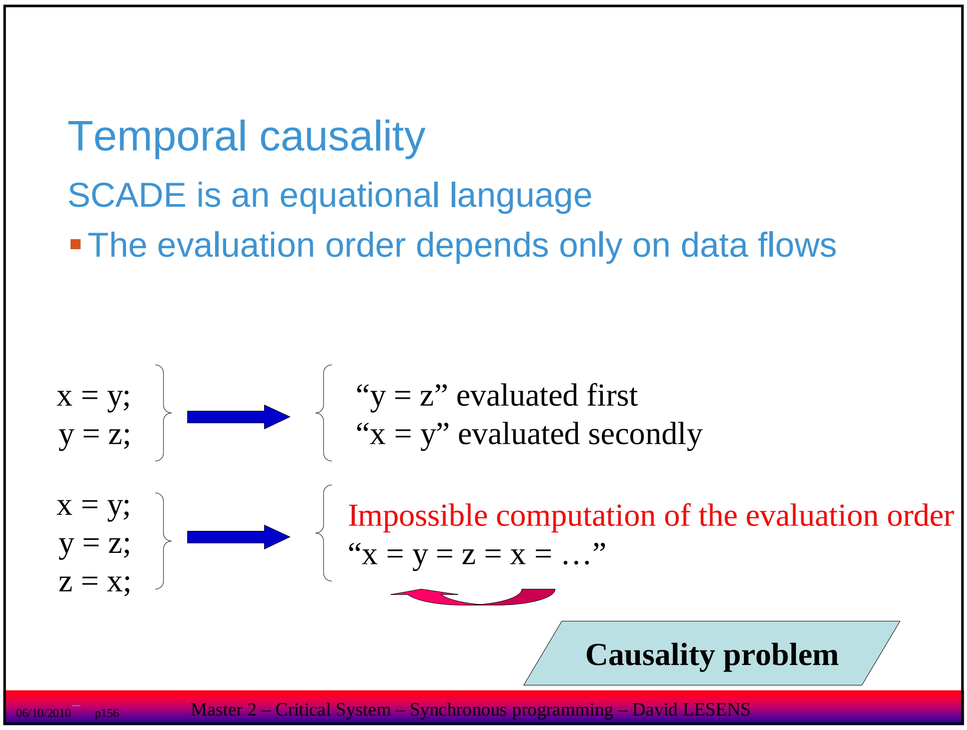
Costly
Non exhaustive



Semantics verification (1/2)

Semantics of a SCADE model

- Syntax
- Typing verification
  - Types compatibility
    - Example: Integer ≠ real
- Non uninitialized variables
- Temporal causality
- …



Temporal causality

SCADE is an equational language

- The evaluation order depends only on data flows

x = y;
y = z;

“y = z” evaluated first
“x = y” evaluated secondly

x = y;
y = z;
z = x;

Impossible computation of the evaluation order
“x = y = z = x = …”

Causality problem

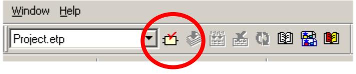
Semantics verification (2/2)

A SCADE model with a correct semantics is:

- Complete
- Consistent
- Implementable

➔ The good properties of a specification

➔ “Semantics check” to be systematically performed



But does the software behave as expected?

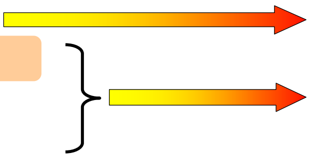
Software validation

Correct software

- No runtime errors
- Satisfaction of real time constraints
- Compliance with the expected results

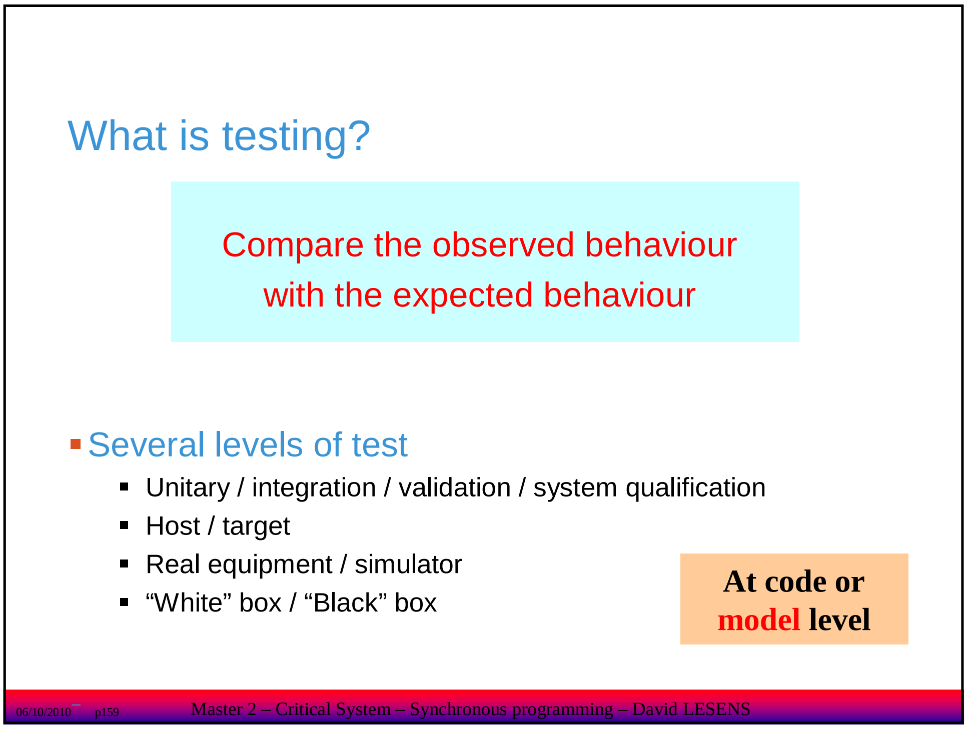
Solutions

- Manual review
- Dynamic testing
  - A code level
  - At model level
- Semantics checking
- Abstract interpretation
- Formal proof



Costly
Error prone

Costly
Non exhaustive

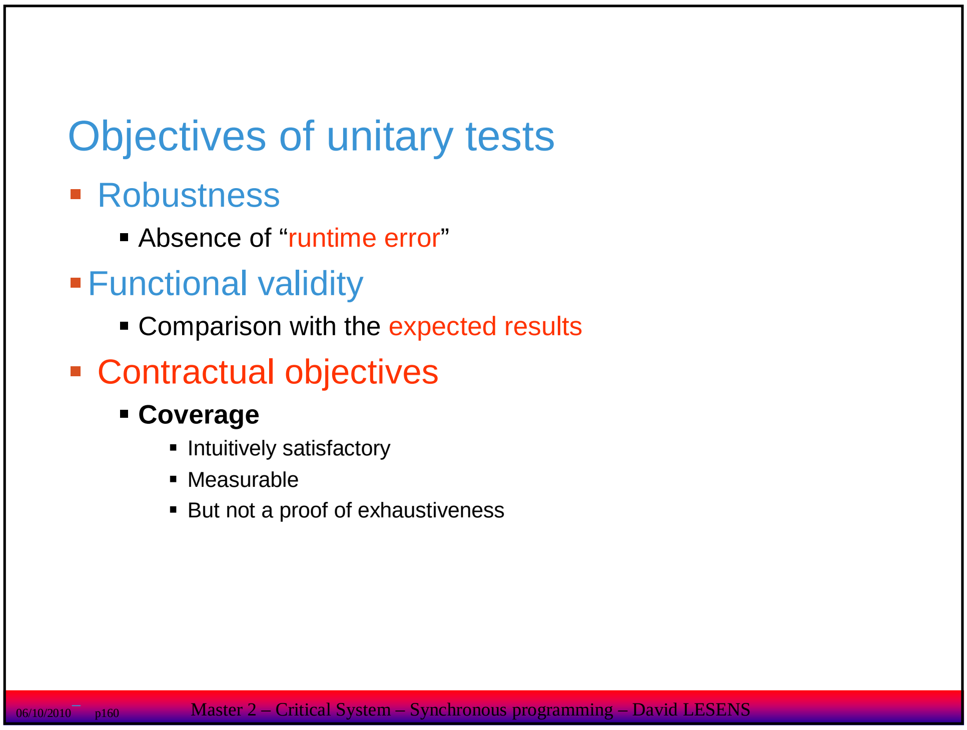


What is testing?

Compare the observed behaviour with the expected behaviour

- Several levels of test
  - Unitary / integration / validation / system qualification
  - Host / target
  - Real equipment / simulator
  - “White” box / “Black” box

At code or model level



Objectives of unitary tests

- Robustness
  - Absence of “runtime error”
- Functional validity
  - Comparison with the expected results
- Contractual objectives
  - **Coverage**
    - Intuitively satisfactory
    - Measurable
    - But not a proof of exhaustiveness

Unitary tests: Coverage

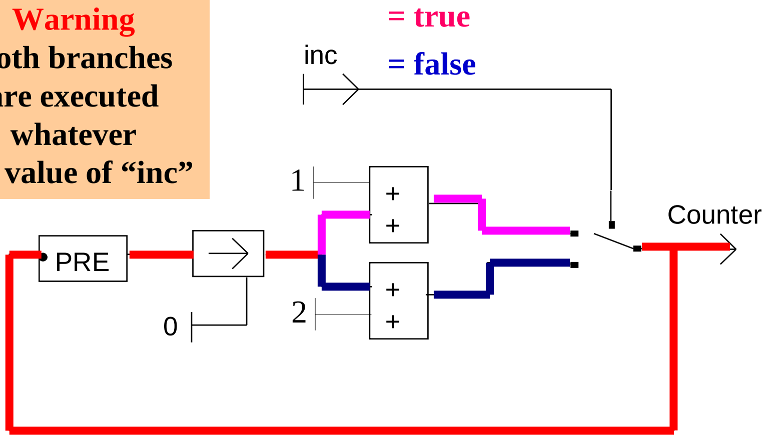
```
Procedure f(x : in real; y: in real; z : out real)
        if (x > 1.0) or (x < -1.0) then
                z := y/x;
        else
                z := y;
        if z < 2.0 then
                z = 2.0;
```

Coverage

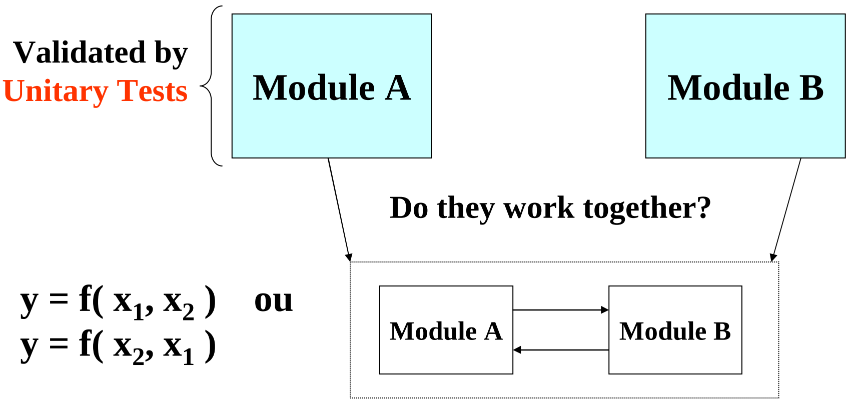
➢ **branch**	**(x=2.0, y=6.0), (x=-1.0,y=1.0)**
➢ **decision**	**+ (x=-2, y=3.0)**
➢ **path**	**+ (x=2.0, y=1.0), (x=0.5,y=2.0)**

Coverage of a SCADE model

Warning
Both branches are executed whatever the value of "inc"



Integration test

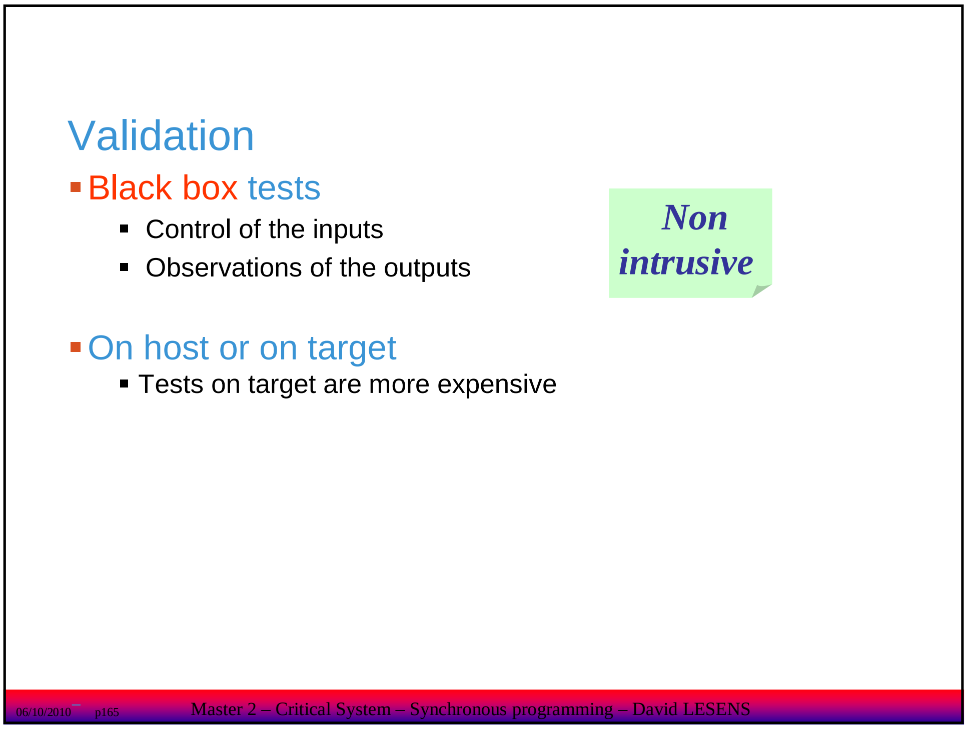


$y = f(x_1, x_2)$ **ou**
$y = f(x_2, x_1)$

Validation of interfaces in white box

Limit of the white box approach

- The presence of a spy may modify the real time behaviour
- What happens if the debugger / simulator has … a bug?

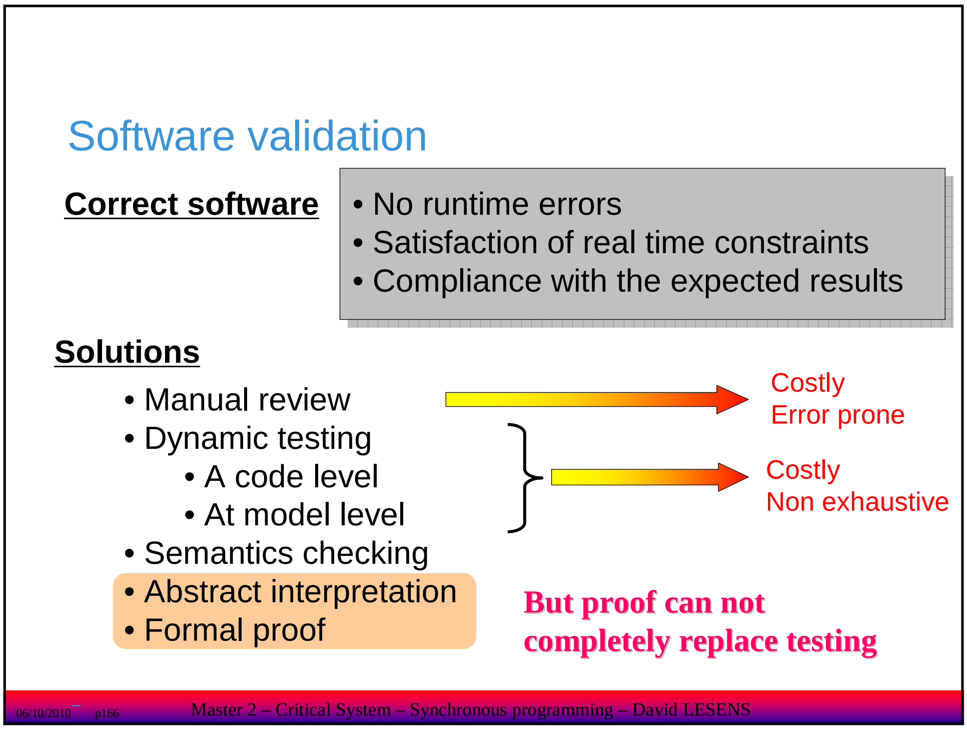


Validation

- Black box tests
  - Control of the inputs
  - Observations of the outputs

Non intrusive

- On host or on target
  - Tests on target are more expensive



Software validation

Correct software

- No runtime errors
- Satisfaction of real time constraints
- Compliance with the expected results

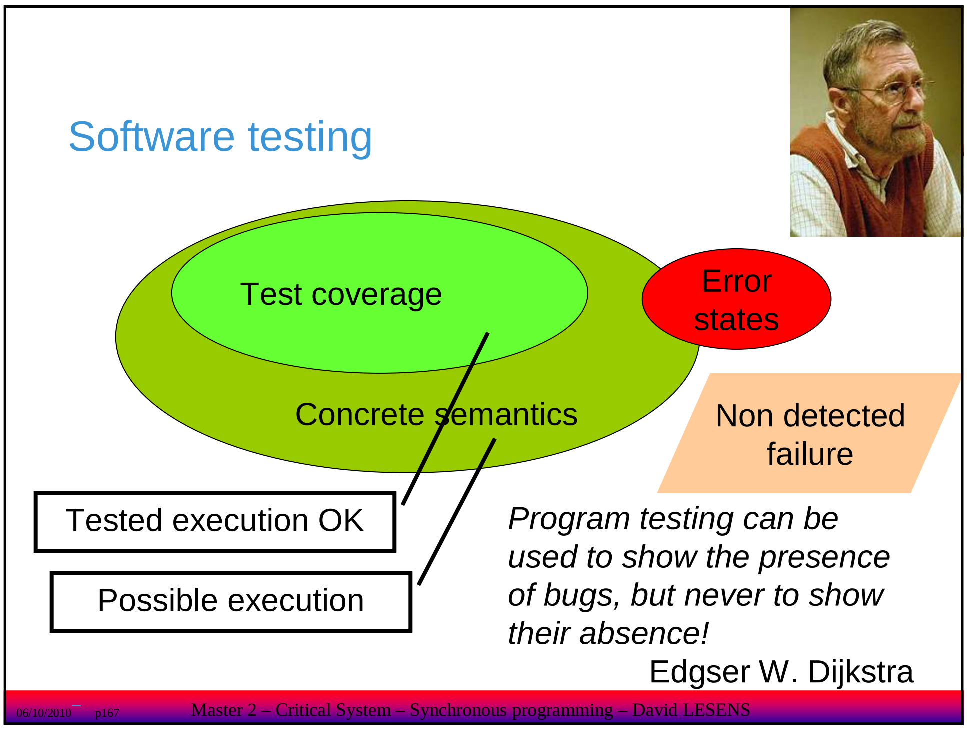
Solutions

- Manual review
- Dynamic testing
  - A code level
  - At model level
- Semantics checking
- Abstract interpretation
- Formal proof

Manual review → Costly, Error prone

Dynamic testing (A code level, At model level) → Costly, Non exhaustive

But proof can not completely replace testing



Software testing

Test coverage

Concrete semantics

Error states

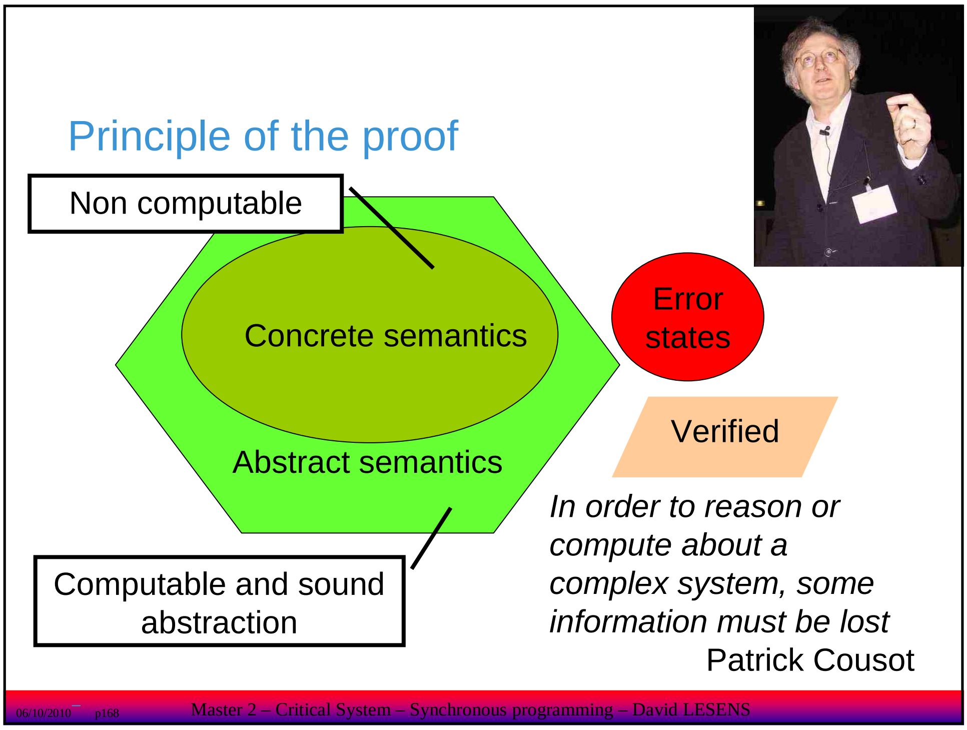
Non detected failure

Tested execution OK

Possible execution

Program testing can be used to show the presence of bugs, but never to show their absence!

Edgser W. Dijkstra



Principle of the proof

Non computable

Concrete semantics

Abstract semantics

Error states

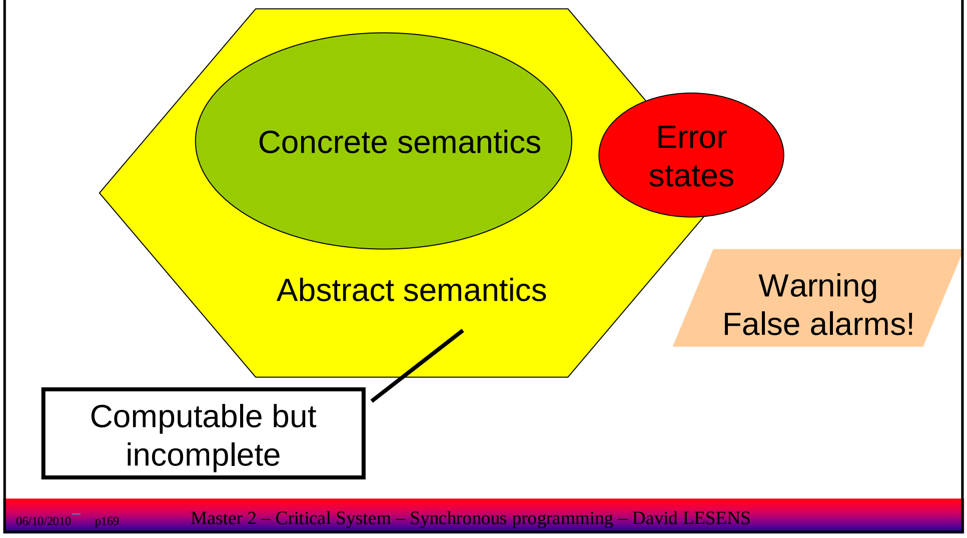
Verified

Computable and sound abstraction

In order to reason or compute about a complex system, some information must be lost

Patrick Cousot

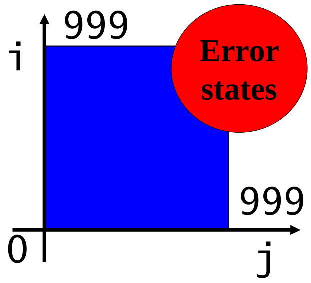
Proof limitation



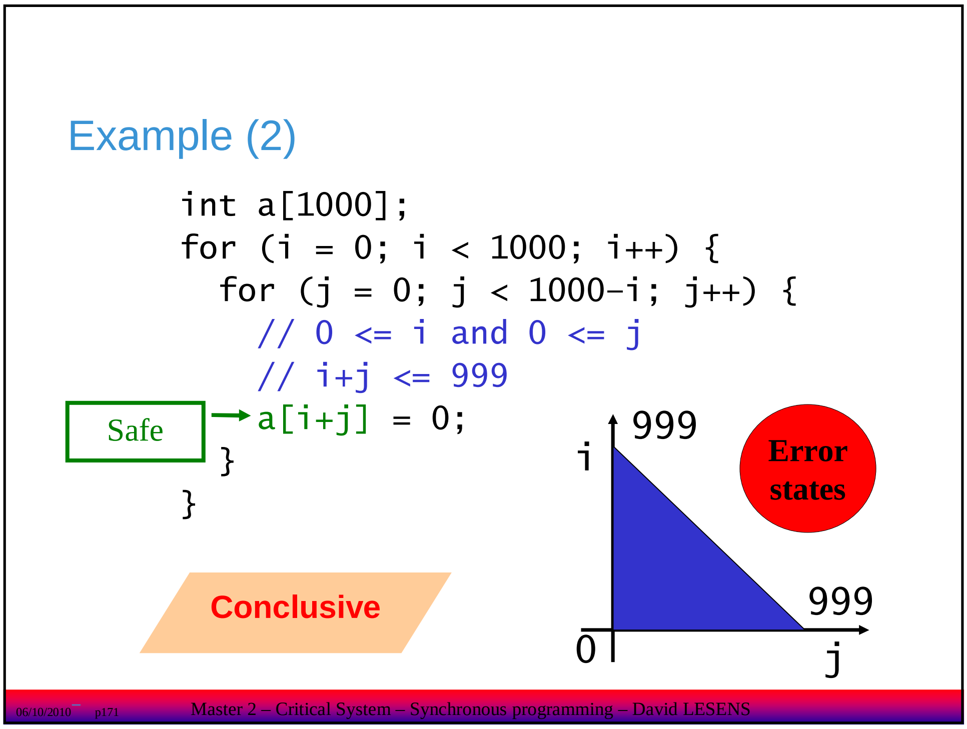
Example (1)

```
int a[1000];
for (i = 0; i < 1000; i++) {
  for (j = 0; j < 1000-i; j++) {
    // 0 <= i <= 999
    // 0 <= j <= 999
    a[i+j] = 0;
  }
}
```

Warning

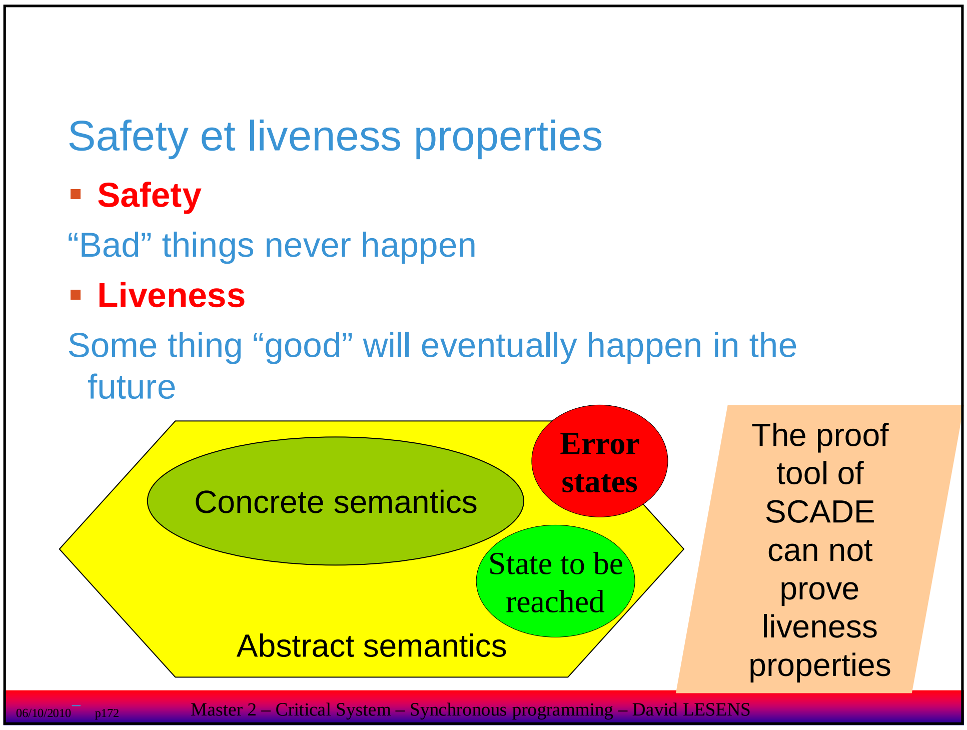


Non conclusive



Example (2)

```
int a[1000];
for (i = 0; i < 1000; i++) {
  for (j = 0; j < 1000-i; j++) {
    // 0 <= i and 0 <= j
    // i+j <= 999
    a[i+j] = 0;
  }
}
```



Safety et liveness properties

- **Safety**

"Bad" things never happen

- **Liveness**

Some thing "good" will eventually happen in the future

The proof tool of SCADE can not prove liveness properties

Interest of the liveness properties

- “Liveness” property / “timed” property
  - Example: if an error is detected, the software shall raise an alarm toward the user
    - Liveness: the alarm will mandatorily be raised (one day or another)

But when?

➔ Not *acceptable for a critical real time piece of software*

❖ Timed property: the alarm will mandatorily be raised 1 second after the failure occurence

➔ **Safety property**

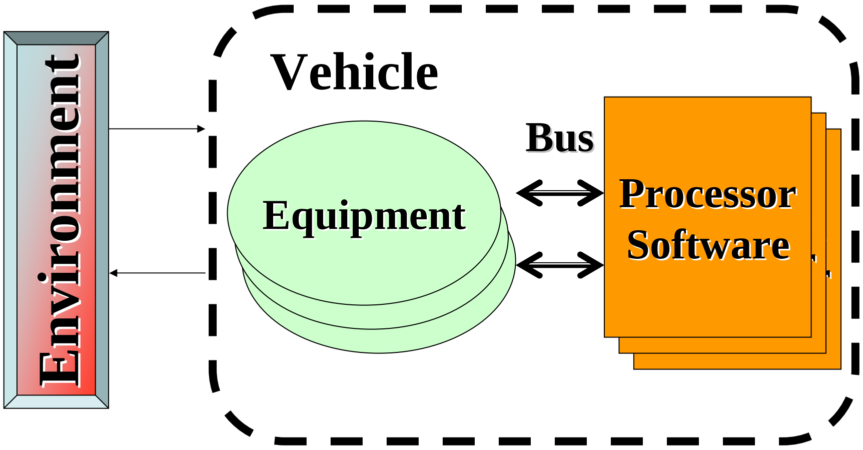
Formal proof

- “Mathematical” exhaustive demonstration that a piece of software/code satisfied a property

➔ Rarely the case!

A piece software generally satisfies a property only in a correct environment

The software is part of a complex system



Formal proof principles

- Software under validation
- Properties to be satisfied
- Software environment

(□ correct environment) ∧ software ⇒ properties

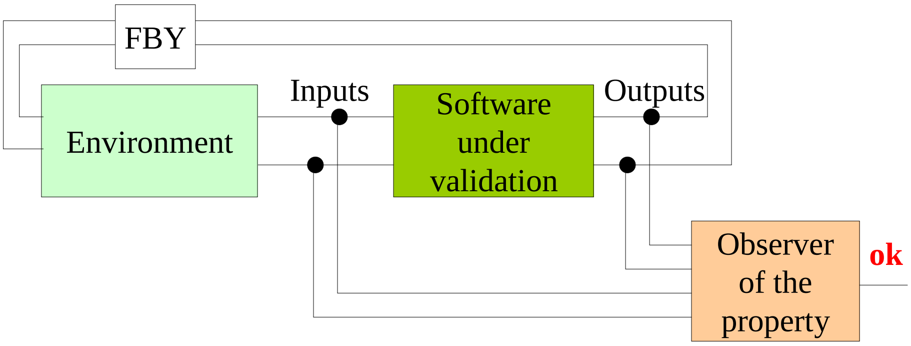
- Environment in open or close loop

Expression of properties

Notion of observer

- An observer is a software observing the software under validation and returning “true” as long as the property is satisfied
  - Observation of the software inputs
  - Observation of the software outputs
- Idem for the environment properties

Observers in SCADE



➔ Use for testing (oracle)
➔ Use by SCADE proof tool

Non deterministic environment (1/2)

The software environment is generally not fully deterministic

- Human action
- Failure
- …

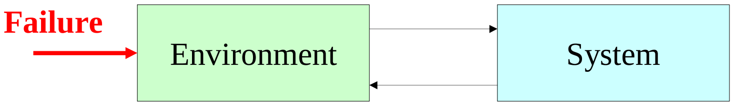
➔ Non deterministic environment

But SCADE is a deterministic language!

Non deterministic environment (2/2)

The non determinism is modelled by an additional input

Example: Failure occurrence



Assertion

An assertion allows to restrict an environment “too much” non deterministic

Example:

- Input “gf” models a gyroscope failure
- Input “tf” models a thruster failureune panne d’une tuyère
  - ➔ To develop a “one fault tolerant” system

Hypothesis: assert #(gf, tf)

The End