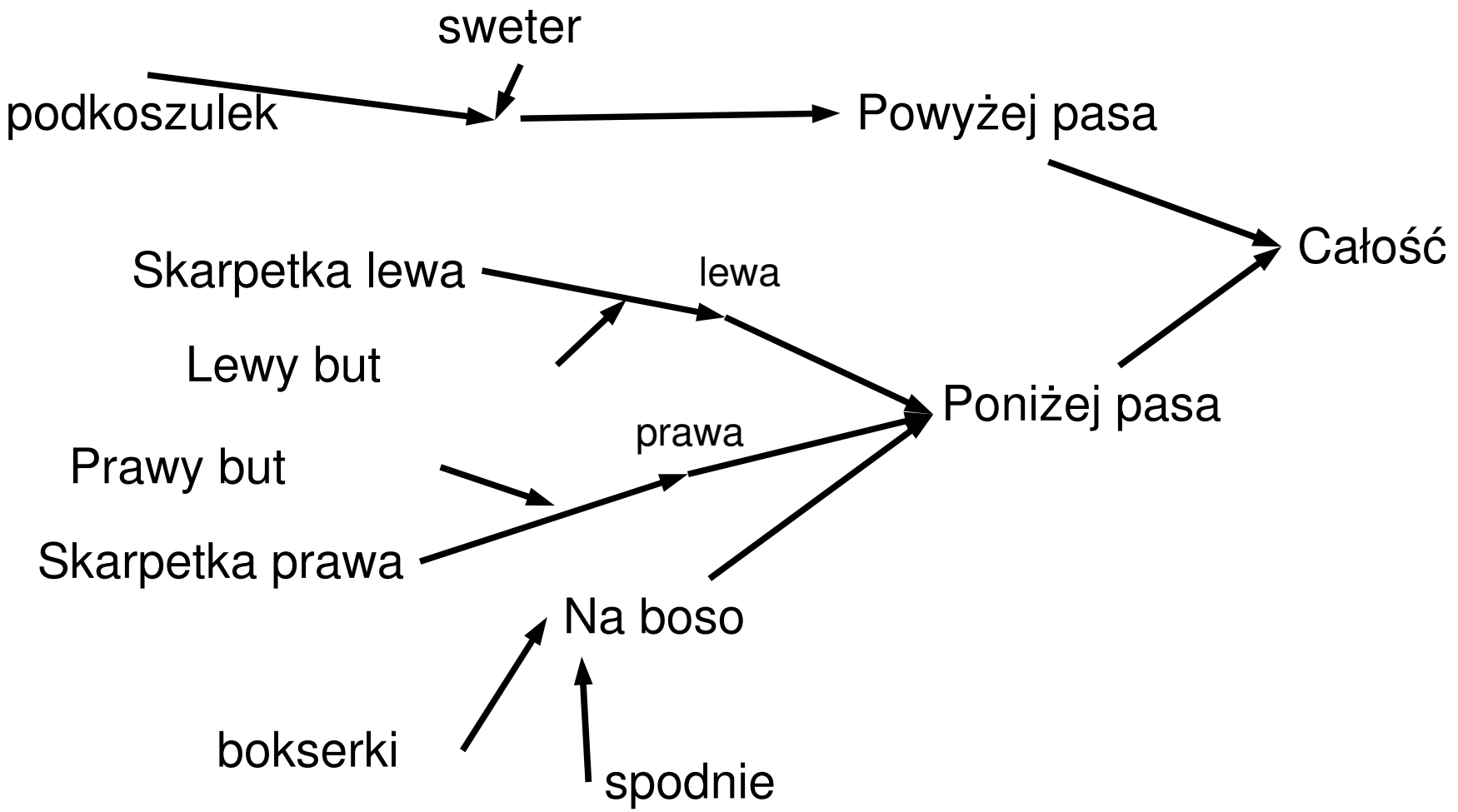
Wprowadzenie do make

Graf zależności



make i plik Makefile

- make działa przede wszystkim na plikach
- Program make jest narzędziem który na podstawie grafu zależności i reguł jest w stanie zbudować pliki lub je odświeżyć (stworzyć nowszą wersje)
- Makefile jest plikiem opisującym graf zależności i reguły dla programu make; i jest indywidualny dla każdego projektu

Reguła (składnia,przykład)

```
CEL: ZALEŻNOŚĆI [; POLECENIE]
<tab>POLECENIE
....
```

```
kot: kot.c kot.h
<tab>gcc -g kot.c -o kot
```

Jak odświeżać

- uruchomienie `make`

  odświeża pierwszą regułę opisaną w Makefile

- make kocurek

  odświeża cel „kocurek” (i wszystko co jest potrzebne dla „kocurek”)

- Odświeżanie jest lawinowe
- make wypisuje co robi
- Jeżeli chcemy żeby nie wypisywał to przed poleceniem w Makefile wstawiamy @
- Jeżeli błąd wykonania polecenia to koniec przetwarzania reguły (chyba że wstawimy „-” przed poleceniem)
- $ jest interpretowany. Jeżeli chcemy użyć $ w poleceniu to trzeba wstawić $$.

```
edit:main.o command.o display.o \
  instert.o search.o files.o

<tab>gcc -o edit main.o command.o \
  display.o instert.o search.o \
  files.o

main.o:main.c defs.h

<tab>gcc -c  main.c

command.o: command.c defs.h \
        commands.h

<tab>@gcc -c command.c

...
```

gcc i make

- Gcc jest w stanie wygenerować Makefile'a

```
gcc -MM lista.c
```

Zaślepki i .PHONY

```
clean:
<tab> rm main.o command.o display.o\
   instert.o search.o files.o edit
```

lepiej:

```
.PHONY:clean
clean:
<tab> rm main.o command.o display.o\
   instert.o search.o files.o edit
```

Zmienne

- W Makefiele'u mogą występować zmienne. Mamy:
  - Referencje

    ```
    obiekty = ala.html gosia.html zosia.html
    obiekty = $(zmienna)
    ```

  - Przypisanie

    ```
    pliki := $(obiekty) ala.html
    ```

  - Dopisanie (definicja przez := (z drobnym różnicą))

    ```
    sciezka += /usr/bin
    ```

- Do wartości zmiennych dostajemy się:
  - **`$(zmienna) ${zmienna}`**

Funkcje

- W Makefile'u można korzystać z funkcji wbudowanych (z własnych też) np.:
  - `$(wildcard *.c)  #rozwiniecie`
  - `$(patsubst %c,%o,$(wildcard *.c))`
    `#podmiana`
  - `$(sort 2a b c)  #sortowanied`
  - ```
    dirs := a b c
    files := $(foreach dir, $(dirs),
       $(wildcard ${dir}/*))    #pętla
    ```
  - `$(if $d, niepuste_d, puste_d) #warunek`
    ```

Domniemana dedukcja / domniemane reguły

- make ma wbudowane często używane wzorce reguł i wystaczy podać CEL i ZALEŻNOŚCI a make już sobie wydedukuje którą standardowy wzorzez reguł zastosować

```
main.o: defs.h

command.o: defs.h commands.h
```

Domniemana Dedukcja 2

- Jeżeli mamy tylko domniemaną dedukcje w Makefile'u to możemy też zrobić cos takiego:

```
obj = a.o b.o c.o
$(obj): defs.h
a.o b.o: command.h
a.o b.o c.o ala.o d.o: cosinnego.h
```

Własne wzorce reguł

- Można stworzyć własną domniemaną regułę poprzez napisane wzorca reguły.
  Wzorzec wygląda tak jak zwykła reguła ale zawiera „%”, który dopasowuje się dla CELU do dowolnego niepustego napisu. W reszcie wzorca % oznacza to dopasowanie.
- Przykład:

asm: $(**`patsubst %.s,%.o,$(obj)`**)

%.s: %.c

<tab>gcc -S $<

Dopasowań może być kilka. Wybór zależy od kolejności.

Zmienne automatyczne

W poleceniach można używać specjalnych zmiennych odwołujących się do dopasowania

- $< nazwa pierwszej zależności
- $@ nazwa celu
- $? odświeżane zależności
- $^ nazawy wszystkich zależności

Statyczne wzorce reguł

CELE:WZORZEC-CELU:WZORZEC-ZALEZNOSCI

<tab>POLECENIE

...

np:

```
obj = foo.o bar.o
$(obj):%.o:%.c def.h
<tab>$(CC) -c $(CFLAGS) $< -o $@
```

Poprawnie jest jeżeli jest tylko jedno dopasowanie!

Warunki

- **ifeq ($(CC), gcc)**
  $(CC) -o bar $(obj) $(for_gcc)
  else
  $(CC) -o bar $(obj) $(no_for_gcc)
  endif
- **ifneq ...**
- **ifdef ZMIENNA**
- **ifndef ZMIENNA**

Rekurencyjny Makefile

```
subsystem:
<tab>cd subdir && $(MAKE)
```

lub

```
subsystem:
<tab>$(MAKE) -C subdir
```

```
export ZMIENNA [op= WARTOSC] #eksport
  zmiennej do środowiska potomnego
```

Więcej o make szukaj w

- `pinfo make`
- `info make`
- `man info`