프로그래밍의 원리 2012 가을 - 실습 5
인터페이스로 프로그래밍해보기, 회로의 확장

서울대학교 프로그래밍 연구실
강동옥, 최민아

2012년 10월 18일

이번 실습의 목적은:

- 지난 숙제의 상태가 없는 회로문제(combinational circuit)를 스위치와 상태가 있는 회로(sequential circuit)로 확장하여, 명령형(imparative) 프로그래밍과 값중심(applicative) 프로그래밍의 차이를 익혀봅니다.

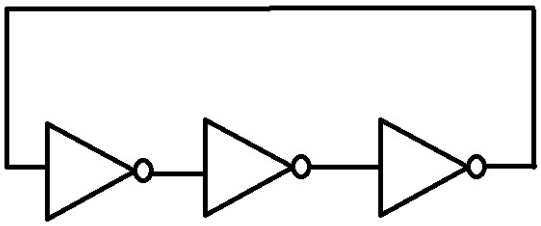
연습문제

명령형 관련 교수님 슬라이드 참고하세요 (링크를 최근 강의록으로 수정하였습니다.)

`http://ropas.snu.ac.kr/~kwang/4190.210/12/3.pdf`

1. 상태가 있는 회로(sequential circuit)

   지난 숙제2에서 만들었던 회로(circuit)는 전기가 한번 흐르고 나면 출력이 하나로 정해집니다. 이런 회로를 상태가 없는 회로(combinational circuit)라고 합니다. 이런 상태가 없는 회로들을 서로가 서로를 맞물리게 연결할 수 있습니다. 그러면 시간에 따라서 전깃줄(wire)의 상태가 계속 변하는 상태 있는 회로(sequential circuit) 가 됩니다. 예를 들어 다음과 같이 NOT 게이트(gate) 세개를 연결하면 각 전깃줄의 전위(voltage)가 0과 1로 반복적으로 변하게 되지요.

   우리는 이번 실습에서 이와같이 게이트의 출력이 다시 입력으로 들어갈 수 있는 회로를 구성할 것입니다. 여기에 더해서 기존과 같이 입력값을 고정하지 않고 사용자가 0과 1을 켜고 끌 수 있는 스위치를 추가 합니다.

(각 게이트의 신호전달시간(propagation time)은 동일하다고 가정. 단위시간이 지날 때마다 게이트의 입력이 출력으로 변하여 나옵니다.)

아래의 함수들을 값중심으로 만드십시요.

(a) 회로를 만들고 사용하는 함수:

`empty-circuit`	:	$circuit$
`add-gate-to-circuit`	:	$gate\text{-}name * gate * circuit \to circuit$
`and-gate`	:	$wire * wire * wire \to gate$
`or-gate`	:	$wire * wire * wire \to gate$
`not-gate`	:	$wire * wire \to gate$
`switch`	:	$wire \to gate$
`match-gate`	:	$wire * circuit \to gate$
`input-wire`	:	$gate * nat \to wire$
`output-wire`	:	$gate \to wire$
`is-and?`	:	$gate \to bool$
`is-or?`	:	$gate \to bool$
`is-not?`	:	$gate \to bool$
`is-switch?`	:	$gate \to bool$

match-gate는 전깃줄과 회로를 받아 인자로 넘겨준 전깃줄이 출력(output wire)인 게이트를 찾아줍니다.

(b) 회로 상태를 만들고 사용하는 함수들:

`set-wire-volts`	:	$initial\text{-}wire\text{-}volts * circuit \to circuit \times state$
`one`	:	$volt$
`zero`	:	$volt$
`step`	:	$circuit \times state \to circuit \times state$
`run`	:	$circuit \times state * nat \to circuit \times state$
`print`	:	$circuit \times state \to void$

프린트 방법은 전깃줄 이름 전위 형식으로 써주세요 예를들어

```
w1 0
w2 1
...
```

*참고

$$
\begin{aligned}
\mathit{initial\text{-}wire\text{-}volt} &= (\mathit{wire} \times \mathit{volt})\ \mathit{list} \\
\mathit{wire} &= \mathit{string} \\
\mathit{switch} &= \mathit{string} \\
\mathit{gate\text{-}name} &= \mathit{string}
\end{aligned}
$$

아래의 함수들을 구현하세요. 전깃줄(wire)에 대해 명령형으로 만드십시요.

(a) 회로를 만들고 사용하는 함수:

$$
\begin{aligned}
\texttt{empty-circuit} &: \mathit{circuit} \\
\texttt{add-gate-to-circuit} &: \mathit{gate\text{-}name} * \mathit{gate} * \mathit{circuit} \to \mathit{circuit} \\
\texttt{one} &: \mathit{volt} \\
\texttt{zero} &: \mathit{volt} \\
\texttt{make-wire} &: \mathit{name} \to \mathit{wire} \\
\texttt{set-wire} &: \mathit{wire} * \mathit{volt} \to \mathit{wire} \\
\texttt{volt-wire} &: \mathit{wire} \to \mathit{volt} \\
\texttt{and-gate} &: \mathit{wire} * \mathit{wire} * \mathit{wire} \to \mathit{gate} \\
\texttt{or-gate} &: \mathit{wire} * \mathit{wire} * \mathit{wire} \to \mathit{gate} \\
\texttt{not-gate} &: \mathit{wire} * \mathit{wire} \to \mathit{gate} \\
\texttt{switch} &: \mathit{wire} \to \mathit{gate} \\
\texttt{match-gate} &: \mathit{wire} * \mathit{circuit} \to \mathit{gate} \\
\texttt{input-wire} &: \mathit{gate} * \mathit{nat} \to \mathit{wire} \\
\texttt{output-wire} &: \mathit{gate} \to \mathit{wire} \\
\texttt{is-and?} &: \mathit{gate} \to \mathit{bool} \\
\texttt{is-or?} &: \mathit{gate} \to \mathit{bool} \\
\texttt{is-not?} &: \mathit{gate} \to \mathit{bool} \\
\texttt{is-switch?} &: \mathit{gate} \to \mathit{bool}
\end{aligned}
$$

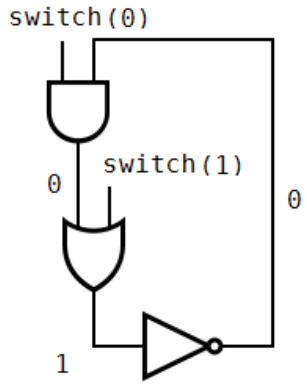
(b) 회로 상태를 만들고 사용하는 함수들:

$$
\begin{aligned}
\texttt{set-wire-volts} &: \mathit{initial\text{-}wire\text{-}volts} * \mathit{circuit} \to \mathit{circuit} \\
\texttt{step} &: \mathit{circuit} \to \mathit{circuit} \\
\texttt{run} &: \mathit{circuit} * \mathit{nat} \to \mathit{circuit} \\
\texttt{print} &: \mathit{circuit} \to \mathit{void}
\end{aligned}
$$

2. 안정된 회로(collecting wire states)

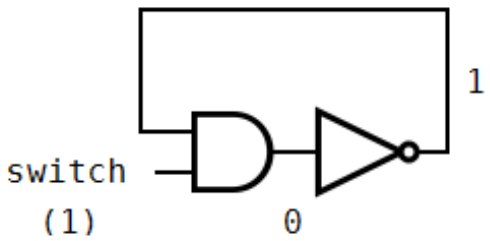
어떤 회로에서 전깃줄이 가질 수 있는 전위들을 알아보고 싶습니다. 즉 어떤 전깃줄이 동작 중에 스위치에 상관 없이 항상 0만 가지는지 혹은 항상 1만 가지는지, 0과 1을 다 가질 수 있는지 체크해보고 싶습니다. 회로 상태에서 스위치 입력은 무시했을때 각 전깃줄마다 저장될수 있는 전위(volt) 집합을 구해봅시다. 값중심 프로그래밍으로만 만들어 봅니다.

$$
\texttt{collect} \;:\; \mathit{circuit} * \mathit{initial\text{-}wire\text{-}volts} \to (\mathit{wire} \times (\mathit{volt}\ \mathit{list}))\ \mathit{list}
$$

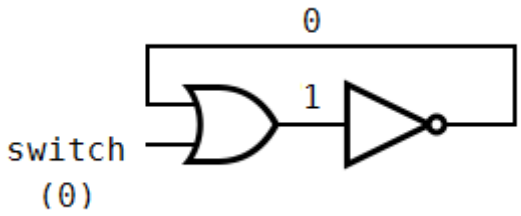



```
(define and1 (and-gate "w1" "w2" "w3"))
(define or1 (or-gate "w3" "w4" "w5"))
(define not1 (not-gate "w5" "w2"))
(define switch1 (switch "w1"))
(define switch2 (switch "w4"))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "or1" or1 circuit1))
(define circuit3 (add-gate-to-circuit "not1" not1 circuit2))
(define circuit4 (add-gate-to-circuit "switch1" switch1 circuit3))
(define circuit5 (add-gate-to-circuit "switch2" switch2 circuit4))
(define circuit-state
  (set-wire-volts
   (list
    (cons "w1" zero) (cons "w2" zero)
    (cons "w3" zero) (cons "w4" one) (cons "w5" one))
   circuit5))
(print (run circuit-state 1))
> w1 0
> w2 0
> w3 0
> w4 1
> w5 1

(collect circuit5
         (list
          (cons "w1" zero) (cons "w2" zero)
```

```
          (cons "w3" zero) (cons "w4" one) (cons "w5" one)))

(define w1 (make-wire "w1"))
(define w2 (make-wire "w2"))
(define w3 (make-wire "w3"))
(define w4 (make-wire "w4"))
(define w5 (make-wire "w5"))
(define and1 (and-gate w1 w2 w3))
(define or1 (or-gate w3 w4 w5))
(define not1 (not-gate w5 w2))
(define switch1 (switch w1))
(define switch2 (switch w4))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "or1" or1 circuit1))
(define circuit3 (add-gate-to-circuit "not1" not1 circuit2))
(define circuit4 (add-gate-to-circuit "switch1" switch1 circuit3))
(define circuit5 (add-gate-to-circuit "switch2" switch2 circuit4))
(define circuit-state
  (set-wire-volts
   (list
    (cons w1 zero) (cons w2 zero)
    (cons w3 zero) (cons w4 one) (cons w5 one))
   circuit5))
(print (run circuit-state 1))
> w1 0
> w2 0
> w3 0
> w4 1
> w5 1
```

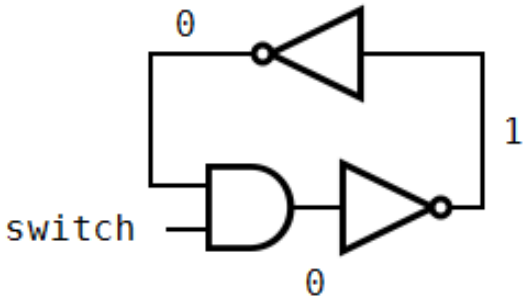


```
(define and1 (and-gate "w1" "w2" "w3"))
(define not1 (not-gate "w3" "w1"))
(define switch1 (switch "w2"))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "switch1" switch1 circuit2))
(define circuit-state
  (set-wire-volts
   (list (cons "w1" one) (cons "w2" one) (cons "w3" zero)) circuit3))
(print (run circuit-state 3))
> w1 0
> w2 1
> w3 0
(collect circuit3 (list (cons "w1" one) (cons "w2" one) (cons "w3" zero)))

(define w1 (make-wire "w1"))
(define w2 (make-wire "w2"))
(define w3 (make-wire "w3"))
(define and1 (and-gate w1 w2 w3))
(define not1 (not-gate w3 w1))
(define switch1 (switch w2))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "switch1" switch1 circuit2))
(define circuit-state
  (set-wire-volts
   (list (cons w1 one) (cons w2 one) (cons w3 zero))
   circuit3))
(print (run circuit-state 3))
> w1 0
> w2 1
> w3 0
```




```
(define or1 (or-gate "w1" "w2" "w3"))
(define not1 (not-gate "w3" "w1"))
(define switch1 (switch "w2"))
(define circuit1 (add-gate-to-circuit "or1" or1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "switch1" switch1 circuit2))
(define circuit-state
  (set-wire-volts
   (list
    (cons "w1" zero) (cons "w2" zero) (cons "w3" one))
   circuit3))
(print (run circuit-state 3))
> w1 1
> w2 0
> w3 1
(collect circuit3 (list (cons "w1" zero) (cons "w2" zero) (cons "w3" one)))

(define w1 (make-wire "w1"))
(define w2 (make-wire "w2"))
(define w3 (make-wire "w3"))
(define or1 (or-gate w1 w2 w3))
(define not1 (not-gate w3 w1))
(define switch1 (switch w2))
(define circuit1 (add-gate-to-circuit "or1" or1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "switch1" switch1 circuit2))
(define circuit-state
  (set-wire-volts
   (list (cons w1 zero) (cons w2 zero) (cons w3 one))
   circuit3))
```

```
> w1 1
> w2 0
> w3 1
```




```
(define and1 (and-gate "w1" "w2" "w3"))
(define not1 (not-gate "w3" "w4"))
(define not2 (not-gate "w4" "w1"))
(define switch1 (switch "w2"))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "not2" not2 citcuit2))
(define circuit4 (add-gate-to-circuit "switch1" switch1 circuit3))
(define circuit-state
  (set-wire-volts
   (list
    (cons "w1" zero) (cons "w2" zero) (cons "w3" zero) (cons "w4" one))
   circuit4))
(print (run circuit-state 1))
> w1 0
> w2 0
> w3 0
> w4 1
(collect circuit4 (list (cons "w1" zero) (cons "w2" zero) (cons "w3" zero)))

(define w1 (make-wire "w1"))
(define w2 (make-wire "w2"))
(define w3 (make-wire "w3"))
(define w4 (make-wire "w4"))
(define and1 (and-gate w1 w2 w3))
```

```
(define not1 (not-gate w3 w4))
(define not2 (not-gate w4 w1))
(define switch1 (switch w2))
(define circuit1 (add-gate-to-circuit "and1" and1 empty-circuit))
(define circuit2 (add-gate-to-circuit "not1" not1 citcuit1))
(define circuit3 (add-gate-to-circuit "not2" not2 citcuit2))
(define circuit4 (add-gate-to-circuit "switch1" switch1 circuit3))
(define circuit-state
  (set-wire-volts
   (list
    (cons w1 zero) (cons w2 zero) (cons w3 zero) (cons w4 one))
   circuit4))
(print (run circuit-state 1))
> w1 0
> w2 0
> w3 0
> w4 1
```