Syntax-directed translation

Rajit Manohar
A direct path from CHP to gates

• Goal: to provide a direct path from CHP to gates
• “Syntax directed”
  ❖ Translation uses the syntax of the CHP program to generate the circuit
  ❖ Uses structural induction
    ‣ Induction on the structure of the program
    ‣ Translations for
      ‣ Base case: assignment, communication, skip, expression evaluation
      ‣ Induction: selections, loops, sequential composition, parallel composition
• History
  ❖ 1980s: multiple approaches developed
  ❖ 1991: Tangram language / Haste @ Handshake Solutions (Philips Research)
  ❖ 1998: Balsa, based on Tangram with extensions (U. Manchester)
Key idea

• Use a communication channel to select a program for execution
• Given a program “P”, we will implement the following

```plaintext
[*  // infinite loop
  [#C];  // wait for pending
  P;    // execute P
  C?    // finish C
]
```

• We execute “P” by simply executing

• This is sometimes called “process decomposition” or “process call”
Wire implementation of channels

- Channel “C” that controls the execution of a program

```
channel
C.r
C.a
```

```
P
```

```
P
```

request
acknowledge

one execution
One execution: idle (waiting) state

Graph showing 'c.r' and 'c.a' signals over time, with 'request' and 'acknowledge' labels. A channel diagram is also present with 'c.r' and 'c.a' inputs leading to a 'wait' state.
One execution: request execution
One execution: running

c.r

request

c.a

acknowledge

channel

c.r

run

P

c.a

Yale ENGINEERING
One execution: done

c.r  

c.a

request

acknowledge

channel

c.r

c.a

done

P

Yale ENGINEERING
One execution: respond to requester
One execution: reset phase

Diagram showing the signal flow with labels for request (c.r) and acknowledge (c.a) over time (x-axis) and different channel states.
One execution: reset phase

- **c.r** (Request)
- **c.a** (Acknowledge)

![Diagram showing the sequence of request and acknowledge signals over time with a channel labeled 'P' and a 'wait' state.](image)
Variables

• Two operations
  ❖ Write a value to the variable
    \[ W!\text{value} \]
  ❖ Read the current value of the variable
    \[ R?x \]

• The variable itself is “passive”
  ❖ It waits for the environment to either write or read its value
Writing and reading a variable

```
write
W.r
W.d[N]
W.a

var

read
R.r
R.d[N]
R.a
```

Yale ENGINEERING
Writing and reading a variable

Write
W.r
W.d[N]
W.a

Read
R.r
R.d[N]
R.a

var

Yale Engineering
Sending and receiving on a channel

```
Send/recv
```

![Diagram showing the process of sending and receiving data on a channel.

The diagram illustrates the interaction between send and recv processes. It shows the flow of data (X.d[N]) and the exchange of control messages (C.r and C.a) over the channel. The graph at the bottom represents the waveform for request and acknowledge signals. The data transmission is depicted with arrows indicating the direction of information flow.]
Expression evaluation

• Example of expression de-composition
Assignment
Assignment

expr \xrightarrow{\text{eval}} \text{xfer} \xrightarrow{\text{write}} \text{var}

expr \xrightarrow{\text{eval}} \text{xfer} \xrightarrow{\text{write}} \text{var}
Building blocks

\[ \text{skip} \]

Yale Engineering
Building blocks

- **skip**
  - c.r
  - c.a

- **S₁ ; S₂**
  - c.r
  - c.a

- **S₁** and **S₂**
  - c.r
  - c.a
Building blocks
Selections and loops

Selection

\[
\begin{align*}
G_1 &\rightarrow S_1 \\
G_2 &\rightarrow S_2
\end{align*}
\]
Selections and loops

Selection

[ \( G_1 \rightarrow S_1 \) ]
[ \( G_2 \rightarrow S_2 \) ]

Loop

*\[ G_1 \rightarrow S_1 \]
*\[ G_2 \rightarrow S_2 \]