`lsc.sty` – Typesetting Live Sequence Charts

Bernd Westphal

February 23, 2006

Contents

1 Introduction

The `lsc.sty` package provides means to typeset Live Sequence Charts. The Live Sequence Charts (LSC) language is a variant of Message Sequence Charts (MSC) [IT99]. It's intended use is as a language to formalise requirements on the communication in a (software or hardware) system under design. The basic difference to MSCs is the addition of modalities to the elements of a chart to distinguish *mandatory* from *possible* behaviour. For example, locations on instance lines can be *hot* to indicate that progress is required to satisfy the LSC (liveness) or *cold* to indicate that behaviour without progress is acceptable. This is where the LSC language gets its name from.

1.1 Supported LSC Language

For further information on the original syntax, semantics, and pragmatics of the LSC language see:

> **[DH01]** Werner Damm and David Harel, *LSCs: Breathing Life into Message Sequence Charts*, Formal Methods in System Design, 19(1):45–80, July 2001.

The LSC language presented in this publication is (up to minor deviations in style) fully supported by `lsc.sty`.

From the original language, two dialects emerged, each of them tailored for a particular use of an LSC specification, i.e. a collection of single charts. The dialect presented in

> **[Klo03]** Jochen Klose, *Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behaviour*, PhD thesis, Carl von Ossietzky Universität Oldenburg, 2003.

considers LSCs as the requirements specification language in a classical development process where the system implementation is finally tested to satisfy the requirements or, possibly better, a system *model* is formally verified to satisfy the requirements.

To this end, the author added timers and timing intervals (also taken from MSCs), local invariants to state requirements on spans of time in contrast to conditions which apply only to single points of time, three activation modes, and two interpretations, but doesn't support scopes. The articles

> **[KW02]** Jochen Klose and Bernd Westphal, *Relating LSC Specifications to UML Models*, In: Hartmut Ehrig and Martin Große-Rhode (Eds.), Proceedings of the Workshop Integration of Software Specification Techniques (INT'02), pages 130–137, April 2002.
>
> **[DW05]** Werner Damm and Bernd Westphal, *Live and Let Die: LSC-based Verification of UML-Models*, Science of of Computer Programming, 55(1–3):117–159, March 2005.

provide a further extension to so called dynamic binding instance lines.

The LSC language presented in these publications is (up to minor deviations in style) fully supported by `lsc.sty`.

The dialect presented in

> **[HM03]** David Harel and Rami Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer-Verlag, 2003.

is tailored for an approach called play-out*play-out*. In this approach, the LSC specification *is* the implementation. A tool called play-engine interprets an LSC specification that has to be related to a GUI. If the GUI is operated by a user, then the play-engine looks up its collection of LSCs and operates the GUI accordingly, thus interacts with a user.

To this end, they added, for example, actor instance lines, actions to modify the system state and extended scopes to loops. They independently also added symbolic instances, means to handle time and to exclude observations by forbidden elements, and two interpretations.

The LSC language presented in [HM03] is not fully supported by `lsc.sty`, and if supported, features are rendered in the style and terminology of [Klo03]. Section 15 discusses how *some* constructs of this LSC dialect can although be typeset with some manual intervention using some special features `lsc.sty`.

1.2 Typesetting MSCs and Sequence Diagrams

As LSCs are a derivative of MSCs as are UML's Sequence Diagrams (SDs), there is a common sublanguage and MSCs or SDs from the common sublanguage may well be typeset using `lsc.sty`.

But note that there are no plans to extend `lsc.sty` to full support of either language. For MSCs, for example, there is an own LaTeX style and SDs may finally be supported by styles for typesetting UML diagrams.

1.3 Related Styles

As soon as `lsc.sty` reached some usable state – of course – it turned out that there exist LaTeX packages for typesetting variants of sequence diagrams from which `lsc.sty` could have been derived from.

We briefly list and discuss the styles we're aware of in the following sections.

1.3.1 MSC

Version 1.13 of the msc`msc.sty` package[1] by S. Mauw and V. Bos [MB01] claims to support the full MSC2000 language.

The most obvious difference is that `msc.sty` seems to be able to compute a layout on its own, i.e. it only takes a description of the dependencies in the LSC, i.e. the user needn't give precise positioning commands while manual intervention is still possible.

In contrast, `lsc.sty` is more low level as it expects *all* layout to be given by the user.

1.3.2 UML

For example, the `pst-uml.sty` package[2] by M. Diamantini provides rudimentary support for UML sequence diagrams.

2 Preliminaries

The macros provided by `lsc.sty` can be divided into three categories:

Environments. The `FullLsc` and `Lsc` environments draw LSC charts with and without header, the `lscinst` and `lsccreateinst` environments draw instances within LSCs, and the `coregion` and `simregion` environments mark coregions and simultaneous regions within instances.

Atomic Commands. An atomic command directly draws an LSC element, instance line segment, or annotation that is related to only a single location. Examples are instance line segments, action boxes, or timer set/reset and timeout.

[1] `http://www.win.tue.nl/ sjouke/misc/mscpackage`
[2] `http://www.ensta.fr/ diam/latex/pst-uml/index.html`

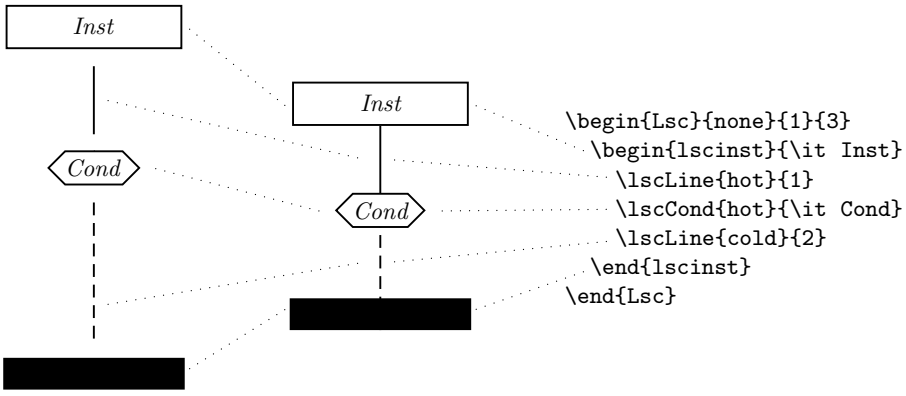


Figure 1: **Exploded view** of a simple LSC.

Composite Commands. Composite commands draw LSC elements that are related to more than one location. There are typically macros to set the locations which are used within an instance line environment and one macro which finally draws the element based on the set locations. A typical example are the macros for messages which comprise one macro for setting the sending location, one macro for setting the reception location, and a macro which draws the message arrow.

Figure 1 illustrates the drawing principle. The LSC in the middle is the result of the source on the right. To pinpoint the relation between commands and graphics, we provide an exploded view on the left and connect graphics and text by dotted lines.

In the example, the `Lsc` environment draws nothing as we use the quantification '`none`' (which is special to `lsc.sty` and not part of any LSC dialect) and have no prechart. If we used '`universal`' or '`existential`', then the `Lsc` would draw the solid or dashed frame around the body and if we had a prechart, `Lsc` would draw the large dashed kandis.[3] But although it doesn't draw something, it defines the `pspicture` environment enclosing the LSC and thereby defines the box boundary. The size of the box is determined by the number of instance lines (1 in the example) and their height (3 `pstricks` units in the example).

Within the `Lsc` environment, there is a single `lscinst` environment. It takes one required argument, the instance name, and is responsible for drawing the instance head and footer. A possible parameter controls the shape of the head and the footer (cf. Section 4).

Within the `lscinst` environment, there are only atomic commands in the example. The `\lscLine` command draws an instance line segment and ad-

[3] A *kandis* is a giant sugar crystal which plays a role in East Frisian tea ceremonies (first the kandis, then the tea, and then slowly and carefully some cream). From the side a kandis typically looks like a hexagon, i.e. like the shape used for rendering conditions and pre-charts. Throughout this text, we use kandis to denote the hexagon shape.

vances vertical position by the segment length. The first parameter is either '`hot`', '`cold`', or '`none`' and gives the temperature (or mode) of the location at the beginning of the instance line segment (cf. Section 5 for the pseudo-mode ''`none`''). The second parameter is the instance line segment length. In the example, we use `pstricks` units, but any length understood by `pstricks` will do (cf. Section 5 for a discussion of environment instance line segments and their optional parameter for timing intervals).

The `\lscCond` command draws a condition restricted to a single instance line, also called narrow condition (cf. Section 7 for a discussion of narrow vs. wide conditions). The first parameter is either '`hot`' or '`cold`' and gives the mode, the second parameter is the condition expression. The `\lscCond` command doesn't advance the vertical position, that is, all atomic commands different from instance line segments which are given between two instance line segments are drawn at the same position and implicitly form a simultaneous region. The `simregion` environment can be used to make this explicit by drawing a large dot at the shared location (cf. Section 13).

Note that the instance line segments in the LSC in the middle of Figure 1 are of different length according to the source on the right but appear to be of roughly the same length. The reason for using different lengths is that the first instance line segment spans from the bottom line of the instance header frame to the top line of the condition kandis. The second instance line segment spans from the top (!) line of the condition kandis to the bottom (!) line of the footer, i.e. the beginning is hidden by the condition and the end is hidden by the footer. The former hiding takes place because only after drawing an instance line segment, all elements in the simultaneous region *before* it are "flushed". The footer is drawn after the last instance line segment has been drawn but it is the only element whose bottom line is positioned at the current position. Conditions, for example, are placed such that the top line lies at the current position.

3 LSC Environments

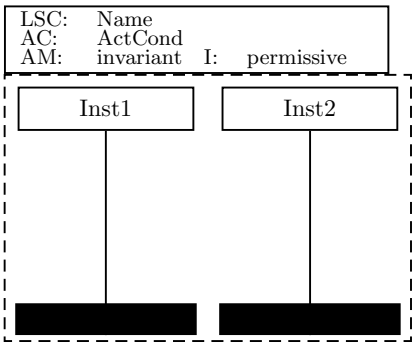
The `lsc.sty` package provides two environments for LSCs, one for complete LSC specifications and one omitting the header, in particular to typeset LSC bodies only without enclosing frame.

3.1 Full LSCs

```
\begin{FullLsc}[prechartloc]{name}{ac}
               {invariant|initial|iterative}
               {strict|weak|permissive|}
               {existential|universal}{nrinsts}{nrlocs}
...
\end{FullLsc}
```

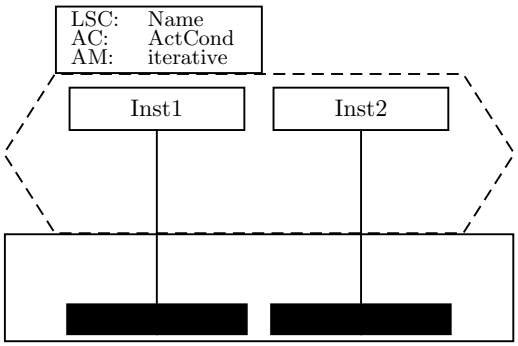
The `Lsc` environment typesets full LSCs, i.e. LSC with a header giving the name *name*, the activation condition *ac*, the activation mode (one of '`invariant`', '`initial`', or '`iterative`'), the interpretation, (one of '`strict`', '`weak`', '`permissive`', or empty), and the quantification (either '`universal`' or '`existential`').

The optional first parameter gives the height of the pre-chart, *nrinsts* is the (strictly positive) number of instances, and *nrlocs* the height of instance lines.




```
\begin{FullLsc}%
        {Name}{ActCond}%
        {invariant}{weak}%
        {existential}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{3}
  \end{lscinst}
\end{FullLsc}
```

Note that the interpretation '`weak`' maps to the unified name permissive in the LSC header. If no interpretation is given, the corresponding field is not present in the LSC header at all.

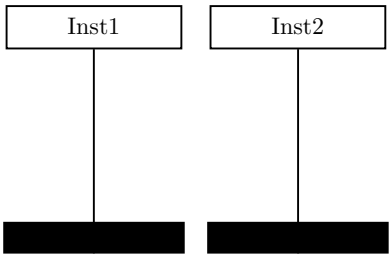



```
\begin{FullLsc}[1.5]%
        {Name}{ActCond}%
        {iterative}{}%
        {universal}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{3}
  \end{lscinst}
\end{FullLsc}
```

3.2 LSCs without Header

```
\begin{Lsc}[prechartloc]
            {existential|universal|none}{nrinsts}{nrlocs}
...
\end{Lsc}
```

The `Lsc` environment is provided to typeset LSCs without header and LSCs without surrounding frame indicating the quantification. With the special quantification '`none`', the frame is omitted and thus only the LSC body drawn. This is useful in publications on LSCs which discuss, for example, the compilation of LSC bodies to automata.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{3}
  \end{lscinst}
\end{Lsc}
```

4 Instance Lines

There are two flavours of instance line environment, ordinary and creation instance lines. Creation instance lines are used to indicate that an object is created during a scenario. To this end the instance head can be placed some offset below the regular position of instance heads and it defines a hook for the composite `\lscCreate` command which draws the creation arrow.

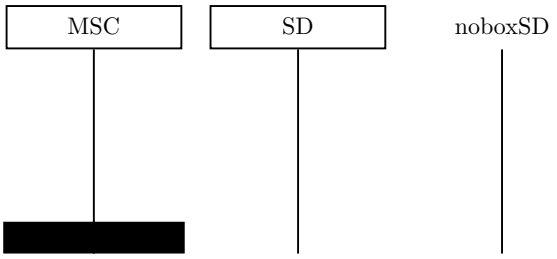
4.1 Ordinary Instance Lines

```
\begin{lscinst}[MSC|SD|noboxSD|esymMSC|
esymSD]{name}
...
\end{lscinst}
```

The *name* is set on top of the instance line segments. The optional parameter controls the appearance of the instance header and footer according to the following table:

	framed header	footer
'MSC' (default)	solid	×
'SD'	solid	–
'noboxSD'	none	–
'esymMSC'	dashed	×
'esymSD'	dashed	–

There is no kind for the fourth possible combination – nobody wants the combination of no frame but a footer.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{MSC}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}[SD]{SD}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}[noboxSD]%
                 {noboxSD}
    \lscLine{hot}{3}
  \end{lscinst}
\end{Lsc}
```

See Section 15.2 for examples of existential symbolic instances.

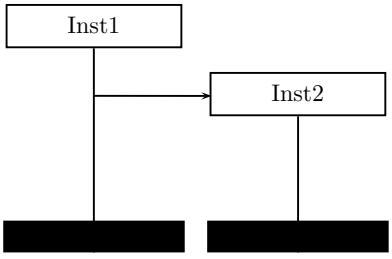
4.2 Creation Instance Lines

```
\begin{lsccreateinst}[MSC | SD | noboxSD | esymMSC |
esymSD]
                          {offset}{id}{name}
...
\end{lsccreateinst}
```

The optional parameter `lsccreateinst` and the *name* work just like in the `lscinst` environment. The additional parameter *offset* gives the offset of the instance header relative to the headers of ordinary instance lines. That is, it is exactly equivalent to `lscinst` if the *offset* is 0 and the corresponding command is omitted.

```
\lscCreate{id}
```

The parameter *id* is a legal LaTeX identifier which can be used in the `\lscCreate` command to draw the create arrow.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{0.7}
    \lscCreate{cr}
    \lscLine{hot}{2.3}
  \end{lscinst}
  \begin{lsccreateinst}{1}{cr}{Inst2}
    \lscLine{hot}{2}
  \end{lsccreateinst}
\end{Lsc}
```

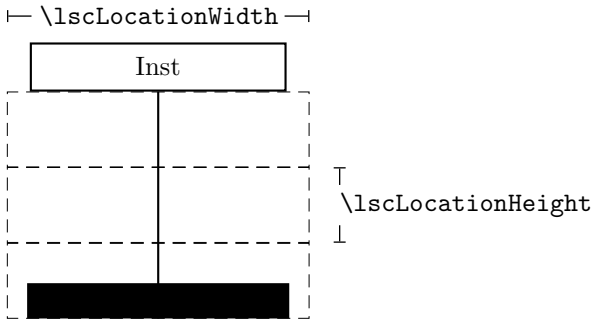
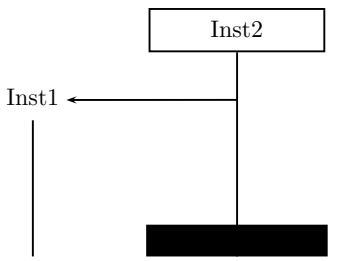


Figure 2: **Dimensions** of a location.




```
\begin{Lsc}{none}{2}{3}
  \begin{lsccreateinst}{1}{cr}{Inst2}
    \lscLine{hot}{2}
  \end{lsccreateinst}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{0.7}
    \lscCreate{cr}
    \lscLine{hot}{2.3}
  \end{lscinst}
\end{Lsc}
```

Note that the creation arrow ends at the border of the frame when the instance head has a frame and at the label if there is no frame. Further note that the instance line end of the arrow is located 0.3 pstricks units above the bottom of the instance head such that the instance line segment lengths on the instance line with the `\lscCreate` can easily be adjusted to obtain a vertical arrow.

The creation arrow is drawn as soon as both ends, i.e. the `\lscCreate` location and the creation instance, are given. The identifier is then undefined, i.e. the same identifier may legally occur in two `\lscCreate/lsccreateinst` pairs in the same LSC – but this is obviously bad style.

4.3 Instance Location and Footer Dimensions

```
\lscLocationWidth
\lscLocationHeight
```

The LSC's box boundary is basically computed in terms of the width and height of a single location within an instance (cf. Figure 2) as defined by lengths `\lscLocationWidth` (default: 3 pstricks units) and `\lscLocationHeight` (default: 1 pstricks unit). The width of the LSC body is computed as

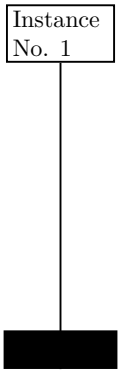
$$\textit{number of instances} * \texttt{\lscLocationWidth}$$

and the height of the LSC body is computed as

$$\textit{number of locations} * \texttt{\lscLocationHeight}.$$

Both lengths can be manipulated, for example, to obtain wider or narrower instances or to globally scale the length of instance line segments.

To change the dimension for only a single LSC, one can exploit that curly braces in LaTeX define scopes and that changes to lengths or commands are local to these scopes. To this end we have enclosed the manipulation of the lengths and the `Lsc` environment in curly braces.



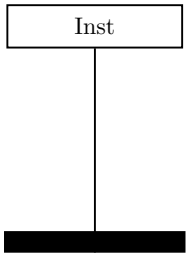

```
{\pssetlength{\lscLocationWidth}{5em}
\pssetlength{\lscLocationHeight}{1.5}
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{%
      \parbox{4em}{Instance No. 1}}
    \lscLine{hot}{3}
  \end{lscinst}
\begin{verbatim}
\end{Lsc}}
```

Note that the height of the instance head frame depends on the height of its content, i.e. a `\parbox` may be used to obtain multiline annotations in instance heads.

`\lscFooterHeightFactor`

The command (!) `\lscFooterHeightFactor` (default: 0.2) affects the appearance of instance footers. It should be set to a positive real number, at best from the left-open interval $]0, 0.2]$. The footer is basically a filled framebox around a `\rule` of height

`\lscFooterHeightFactor` $*$ `\lscLocationHeight`.




```
{\renewcommand{\lscFooterHeightFactor}{0.01}
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst}
    \lscLine{hot}{3}
  \end{lscinst}
\begin{verbatim}
\end{Lsc}}
```

5 Instance Line Segments

There are two flavours of instance line segments, ordinary and environment instance line segments. Environment instance line segments are rendered like the ones of Sequence Diagrams.

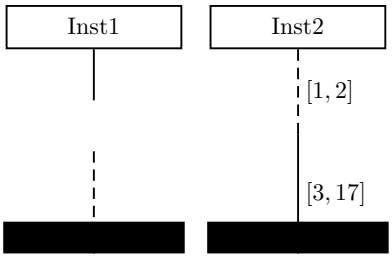
5.1 Ordinary Instance Line Segments

`\lscLine[`*interval*`]{hot|cold|none}{`*length*`}`

The `\lscLine` command draws the elements belonging to the location where the line starts, draws the instance line segment of length *length*, and advances the vertical position by this amount. The mode "`none`" is not a valid mode of LSCs but only present to draw an invisible line segment to draw abstract LSCs, e.g. to explain LSC construction algorithms.

The actual length of the line segment is computed as *length*·`\lscLocationHeight`.

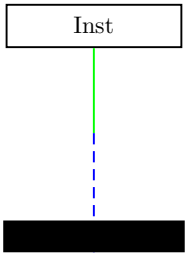
If the optional *interval* is given, it rendered to the right of the instance line segment. This may be any content, but it is typically used to place timing intervals.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{0.75}
    \lscLine{none}{0.75}
    \lscLine{cold}{1.5}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine[{$[1,2]$}]{cold}{1.25}
    \lscLine[{$[3,17]$}]{hot}{1.75}
  \end{lscinst}
\end{Lsc}
```

Note that timing intervals are always vertically centred on the line and put to the right. As this is already the optional parameter, there are no plans to support variable positioning of the timing interval but nobody keeps you from using `\rput` should the need arise.

The `\lscLine` command doesn't set the line colour, thus one can obtain colourful LSCs using the `\psset` command.

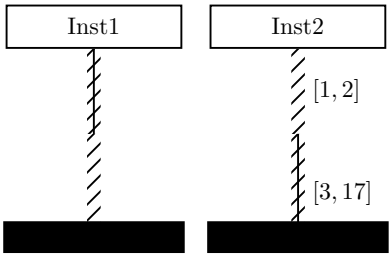



```
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst}
    {\psset{linecolor=green}
    \lscLine{hot}{1.25}}
    {\psset{linecolor=blue}
    \lscLine{cold}{1.75}}
  \end{lscinst}
\end{Lsc}
```

5.2 Environment Instance Line Segments

`\lscEnvLine[`*interval*`]{hot|cold}{`*length*`}`

The `\lscEnvLine` behaves just like `\lscLine`, only the instance line segment is rendered differently, namely in Sequence Diagram style. To indicate hot segments, a solid line is drawn on top of the environment line segment.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscEnvLine{hot}{1.25}
    \lscEnvLine{cold}{1.75}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscEnvLine[{$[1,2]$}]{cold}{1.25}
    \lscEnvLine[{$[3,17]$}]{hot}{1.75}
  \end{lscinst}
\end{Lsc}
```

6 Messages

Both message kinds distinguished by [Klo03] are supported by two sets of composite commands. For each kind, there is one command to mark the sending location, one command to mark the reception location, and one command to finally draw the arrow.

The difference between both is only the shape of the arrow, it is not enforced that instantaneous messages are drawn horizontally without slope.

Both also don't define the line colour, i.e. one can obtain coloured arrows using `\psset` (cf. Section 5.1).

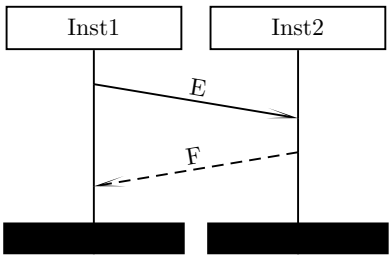
Self messages are currently not supported. If the sending and reception lie on the same instance line, then the arrow's line coincides with the instance line and only the arrow itself and the annotation will be visible.

6.1 Asynchronous Messages

`\lscAsynSnd{`*id*`}`
`\lscAsynRcv{`*id*`}`
`\lscAsyn{`*id*`}{hot` | `cold}{`*label*`}`

If `\lscAsynSnd` and `\lscAsynRcv` occurred exactly once within an `FullLsc` or `Lsc` environment, then a following `\lscAsyn` command with the same *id* draws the arrow annotated with *label*.

The *id* can be reused for another message within the same LSC after `\lscAsyn` has been called, but this is not considered good style.



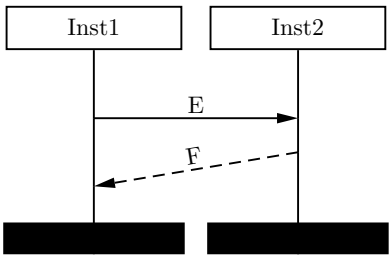

```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{0.5}%
    \lscAsynSnd{e}%
    \lscLine{hot}{1.5}%
    \lscAsynRcv{f}%
    \lscLine{hot}{1}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{1}%
    \lscAsynRcv{e}%
    \lscLine{hot}{0.5}%
    \lscAsynSnd{f}%
    \lscLine{hot}{1.5}%
  \end{lscinst}
  \lscAsyn{e}{hot}{E}%
  \lscAsyn{f}{cold}{F}%
\end{Lsc}}
```

Note that the closing commands of composite commands, like `\lscAsyn` in the example, are by convention collected below all instance environments. It is by no means necessary, as explained above, but thereby one provides a common place to look for all definitions.

6.2 Instantaneous Messages

```
\lscInstSnd{id}
\lscInstRcv{id}
\lscInst{id}{hot | cold}{label}
```

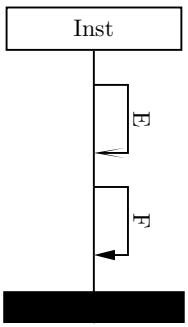
The composite command for instantaneous messages is operated exactly like `\lscAsyn` and friends (cf. Section 6.1).




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}%
    \lscInstSnd{e}%
    \lscLine{hot}{1}%
    \lscInstRcv{f}%
    \lscLine{hot}{1}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{1}%
    \lscInstRcv{e}%
    \lscLine{hot}{0.5}%
    \lscInstSnd{f}%
    \lscLine{hot}{1.5}%
  \end{lscinst}
  \lscInst{e}{hot}{E}%
  \lscInst{f}{cold}{F}%
\end{Lsc}}
```

6.3 Self Messages

Self messages are not a third category but both asynchronous and instantaneous messages may well begin and end at the same instance line.




```
\begin{Lsc}{none}{1}{4}
  \begin{lscinst}{Inst}
    \lscLine{hot}{0.5}
    \lscAsynSnd{e}
    \lscLine{hot}{1}
    \lscAsynRcv{e}
    \lscLine{hot}{0.5}
    \lscInstSnd{f}
    \lscLine{hot}{1}
    \lscInstRcv{f}
  \end{lscinst}
  \lscAsyn{e}{hot}{E}
  \lscInst{f}{hot}{F}
\end{Lsc}
```

Currently, the shape is fixed, i.e. the arrow extends to the right of the instance line and has “armlength” 0.5 pstricks units.

7 Conditions

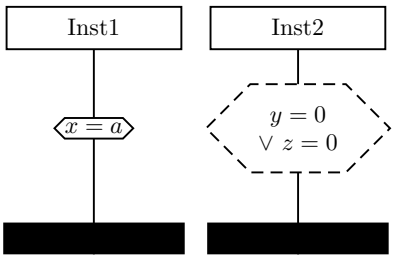
Conditions can be rendered using an atomic command and a composite command. The former command is provided for convenience for the frequent case that a condition spans only a single instance line (narrow condition). In this case, all information, i.e. the location, the temperature, and the label, can be given immediately.

A wide condition may also span only one instance line but it may also cover more.

7.1 Narrow Conditions

`\lscCond{hot|cold}{`*label*`}`

The `\lscCond` command renders a condition kandis at the current position.



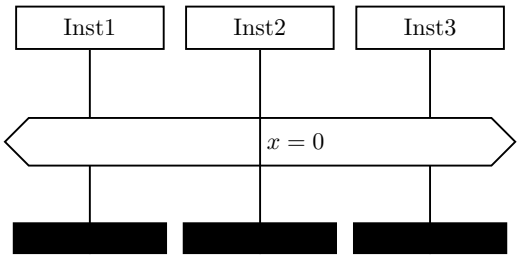

```
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}%
    \lscCond{hot}{$x = a$}%
    \lscLine{hot}{2}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{0.5}%
    \lscCond{cold}{%
      \parbox{4em}{\centering%
        $y=0$ $\vee\ z=0$}}%
    \lscLine{hot}{2.5}%
  \end{lscinst}
\end{Lsc}}
```

The size of the kandis scales with the size of *label*'s box. This has the effect that it is rather low if *label* has neither over- nor underlength and it scales unproportionally if *label* has multiple lines. The former issue can be addressed using `\rule` or `\phantom`, the latter will have to be addressed by a smarter kandis drawing macro within `lsc.sty`.

7.2 Wide Conditions

`\lscWidecondOn{`*id*`}`
`\lscWidecondOff{`*id*`}`
`\lscWidecond{`*id*`}{hot|cold}{`*label*`}`

The `\lscWidecondOn` and `\lscWidecondOff` commands add locations to a wide condition. They have to occur on exactly the same height and consecutively. Use `\lscWidecondOn` for those instances to which the condition applies and `\lscWidecondOff` otherwise.




```
\pssetlength%
  {\lscLocationWidth}{2.5}%
\begin{Lsc}{none}{3}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}
    \lscWidecondOn{wc}
    \lscLine{hot}{2}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{1}
    \lscWidecondOff{wc}
    \lscLine{hot}{2}
  \end{lscinst}
  \begin{lscinst}{Inst3}
    \lscLine{hot}{1}
    \lscWidecondOn{wc}
    \lscLine{hot}{2}
  \end{lscinst}
  \lscWidecond{wc}{hot}%
    {$\qquad\quad x = 0$}
\end{Lsc}
```

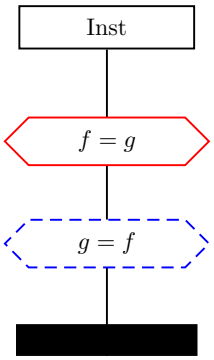
The *label* is set in the centre of the kandis. If the condition doesn't apply to the central location, one may need to add some offset like in the example above to avoid the instance line segment running through the label.

Note that the size of wide conditions is fixed, i.e. it does in particular not scale with the height of *label*'s box. As a consequence, *label* should fit onto a single line.

7.3 Colourful Conditions

The appearance of narrow conditions, and in particular the colour of its outline, cannot be easily manipulated because narrow conditions are "scheduled" and drawn when the next instance line segment is drawn (cf. Section 13). Thus changing the colour of the outline applies to all elements which are drawn together with the instance line segment, in particular the instance line itself.

If wide conditions are sufficient (cf. Section 7.2), their colour can be modified as shown in the following example.




```
\let\lscWidecondOooginool\lscWidecond%
\renewcommand{\lscWidecond}[3]{{%
  \ifthenelse{\equal{#2}{hot}}{%
    \psset{linecolor=red}}{%
    \psset{linecolor=blue}}%
  \lscWidecondOooginool{#1}{#2}{#3}}}
\begin{Lsc}{none}{1}{4.5}
  \begin{lscinst}{Inst}
    \lscLine{hot}{1}%
    \lscWidecondOn{c}%
    \lscLine{hot}{1.5}%
    \lscWidecondOn{d}%
    \lscLine{hot}{2}%
  \end{lscinst}
  \lscWidecond{c}{hot}{$f = g$}%
  \lscWidecond{d}{cold}{$g = f$}%
\end{Lsc}
```

8 Local Invariants

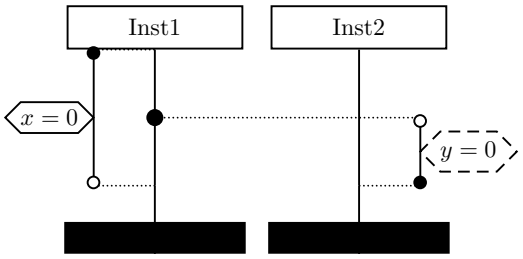
```
\lscLocinvStart{id}
\lscLocinvBegin{id}
\lscLocinvEnd{id}
\lscLocinv[r|l]{id}{incl|excl}{incl|excl}{hot|cold}{label}
```

The commands `\lscLocinvBegin` (or, synonymously, `\lscLocinvStart`) and `\lscLocinvEnd` mark the beginning and end position of local invariant *id*.

The `\lscLocinv` command renders the local invariant *id*. The position of the vertical line to which the kandis is connected can be controlled by the optional parameter (default: '`r`' (to the right)).

The third and fourth parameters select whether the ends are inclusive ('`incl`') or exclusive ('`excl`'). Inclusive ends are rendered with a filled circle and exclusive ends with a non-filled circle (just like jumps are indicated in function graphs). This is different from [Klo03] where the kandis is rotated by 90 degrees and inclusive ends are indicated by having the kandis end in a rectangular shape instead of triangular. The drawback is that a both-inclusive local invariant looks like a rotated action box and doesn't resemble conditions at all.[4]

[4] And – first of all – it seemed easier to render the shape we chose.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLocinvBegin{li}
    \lscLine{hot}{1}
    \begin{simregion}
      \lscLocinvStart{lj}
    \end{simregion}
    \lscLine{hot}{1}
    \lscLocinvEnd{li}
    \lscLine{hot}{1}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{2}
    \lscLocinvEnd{lj}
    \lscLine{hot}{1}
  \end{lscinst}
  \lscLocinv[l]{li}%
    {incl}{excl}{hot}{$x = 0$}
  \lscLocinv{lj}%
    {excl}{incl}{cold}{$y = 0$}
\end{Lsc}
```

In the example above, we used a simregion (cf. Section 13) to highlight the point where the local invariant connects to the instance line in lack of other elements. Local invariants in the wild typically end at simultaneous regions with other elements as the LSC's automaton becomes non-deterministic otherwise.

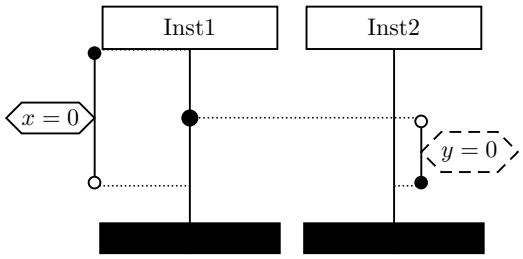
Note that the local invariant kandis behaves like that of narrow conditions. That is, it looks too low if *label* has no over- or underlength and it scales unproportionally with multiline *label*s (cf. Section 7.1).

Further note that the box boundary of the LSC doesn't consider the amounts by which local invariants kandis stand out to the left or the right. One has to explicitly treat kandis overlapping other text by adding vertical space to the left or the right.[5]

`\lscLocinvOffset`

The length `\lscLocinvOffset` (default: 0) is added to the horizontal distance between instance line and local invariant. To move the local invariant further outwards, set it to a positive number and to move it further inwards, set it to a negative number.

[5] Technically the problem is that the box boundary is determined when the `Lsc` environment is opened. At this point in time the extension of local invariants is not yet known. And the dimension cannot be changed afterwards (as far as the author knows).




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLocinvBegin{li}
    \lscLine{hot}{1}
    \begin{simregion}
      \lscLocinvStart{lj}
    \end{simregion}
    \lscLine{hot}{1}
    \lscLocinvEnd{li}
    \lscLine{hot}{1}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{2}
    \lscLocinvEnd{lj}
    \lscLine{hot}{1}
  \end{lscinst}
  \pssetlength{\lscLocinvOffset}{0.5}
  \lscLocinv[l]{li}%
    {incl}{excl}{hot}{$x = 0$}
  \pssetlength{\lscLocinvOffset}{-0.5}
  \lscLocinv{lj}%
    {excl}{incl}{cold}{$y = 0$}
\end{Lsc}
```

The vertical position of the kandis is fixed. It is always drawn centred between the beginning and end point.

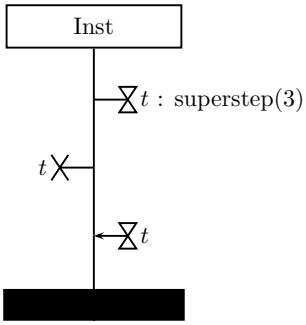
9 Timer Sets, Timer Resets, and Timeouts

`\lscTimerSet[`r`|`l`]{`*label*`}{`*duration*`}`
`\lscTimerReset[`r`|`l`]{`*label*`}`
`\lscTimeout[`r`|`l`]{`*label*`}`

The atomic commands `\lscTimerSet`, `\lscTimerReset`, and `\lscTimeout` render the timer symbols. The position of the hourglass symbols can be controlled by the optional parameter (default: '`r`' (to the right)).

The parameter *label* is not an identifier as used with composite commands but arbitrary text. To connect, e.g., timer setting and timeout they should obtain the same *label*.

A timer type is currently not explicitly supported. It may be written as part of the *label*.

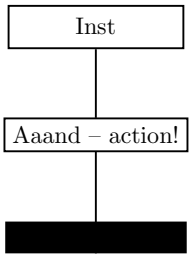



```
\begin{Lsc}{none}{1}{4}
  \begin{lscinst}{Inst}
    \lscLine{hot}{0.75}
    \lscTimerSet{$t$ : superstep}{3}
    \lscLine{hot}{1}
    \lscTimerReset[l]{$t$}
    \lscLine{hot}{1}
    \lscTimeout{$t$}
    \lscLine{hot}{1.25}
  \end{lscinst}
\end{Lsc}
```

10 Actions

`\lscAct{`*label*`}`

The atomic command draws a rectangular action symbol annotated with *label*.



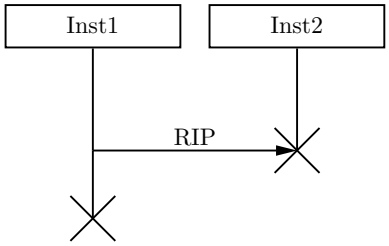

```
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst}
    \lscLine{hot}{1}
    \lscAct{Aaand -- action!}
    \lscLine{hot}{2}
  \end{lscinst}
\end{Lsc}
```

11 Self Destructions

`\lscKill`

The atomic `\lscKill` command draws a cross at the current position to indicate termination of the instance. It is typically not followed by any more `\lscLine` commands.

There is no explicit deletion arrow (like there is a creation arrow) since instantaneous messages may be used for this purpose.

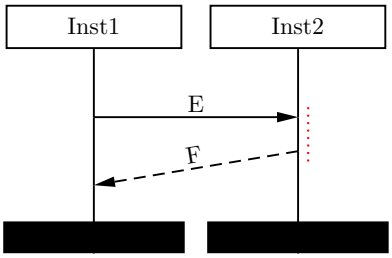



```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}[SD]{Inst1}
    \lscLine{hot}{1.5}
    \lscInstSnd{rip}
    \lscLine{hot}{1}
    \lscKill
  \end{lscinst}
  \begin{lscinst}[SD]{Inst2}
    \lscLine{hot}{1.5}
    \lscInstRcv{rip}
    \lscKill
  \end{lscinst}
  \lscInst{rip}{hot}{RIP}
\end{Lsc}
```

12 Coregions

`\begin{coregion}[l|r]...\end{coregion}`

The elements and instance line segments enclosed in a `coregion` environment are marked with a parallel dotted line to indicate that they form a coregion. The optional parameter controls whether the line appears to the left ('`l`', the default) or to the right ('`r`') of the instance line. `coregion` environments shall not be nested.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}\lscInstSnd{e}%
    \lscLine{hot}{1}\lscInstRcv{f}%
    \lscLine{hot}{1}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{1}%
    \begin{coregion}[r]
      \lscInstRcv{e}%
      \lscLine{hot}{0.5}%
      \lscInstSnd{f}
      \psset{linecolor=red}
    \end{coregion}
    \lscLine{hot}{1.5}%
  \end{lscinst}
  \lscInst{e}{hot}{E}%
  \lscInst{f}{cold}{F}%
\end{Lsc}}
```

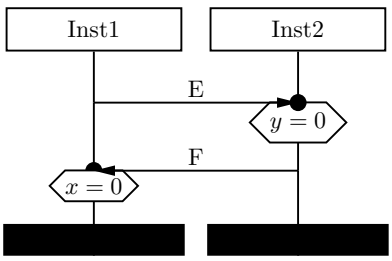
The `coregion` environment doesn't set the line's colour so it can be manipulated using `\psset`. But it does set the width to ensure that the dotted line is noticeable. To change the width one may temporary redefine the internal length `\lsc@coregwidth`.

The coregion line is drawn when the environment is closed. This has the (not so nice) effect that the line lies above condition kandis, action boxes etc. So currently coregions look best when applied to message ends.

13 Simultaneous Regions

`\begin{simregion}[b|f]...\end{simregion}`

The elements enclosed in a `simregion` environment are marked with a large dot to explicitly indicate that they lie in a simultaneous region. The optional parameter controls whether the dot appears behind ('`b`', the default) or in front of ('`f`') the elements. `simregion` environments shall not be nested and shall not comprise instance line segments.




```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{0.75}
    \lscInstSnd{e}
    \lscLine{hot}{1}
    \begin{simregion}
      \lscInstRcv{f}
      \lscCond{hot}{$x=0$}
    \end{simregion}
    \lscLine{hot}{1.25}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{0.75}
    \begin{simregion}[f]
      \lscInstRcv{e}
      \lscCond{hot}{$y=0$}
    \end{simregion}
    \lscLine{hot}{1}
    \lscInstSnd{f}
    \lscLine{hot}{1.25}
  \end{lscinst}
  \lscInst{e}{hot}{E}
  \lscInst{f}{hot}{F}
\end{Lsc}
```

Note that the order of elements (except for cuts) of different kinds in a simultaneous region is not significant. The drawing order is (from bottom to top):

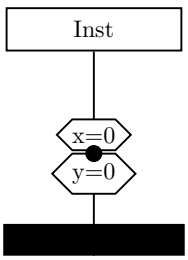
1. simultaneous region dot, if it should go into the background
2. white background of wide condition behind the instance line (`\lscWidecondOn`)
3. white background of wide condition in front of the instance line (`\lscWidecondOff`)
4. action or narrow condition

5. timer set, reset, or timeout

6. atom (cf. Section 14.1)

7. simultaneous region dot, if it should go into the foreground

Per simregion, at most one action or narrow condition can be drawn. If one really needs to put one above the other (typically sacrificing readability) one may use instance line segments of length 0. Two actions or narrow conditions can also be stacked one above the other using a manually tailored instance line segment in between and a foreground simregion circle like in the following example.




```
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst}
    \lscLine{hot}{1}
    \lscCond{hot}{x=0}
    \lscLine{hot}{0.5}
    \begin{simregion}[f]
      \lscCond{hot}{y=0}
    \end{simregion}
    \lscLine{hot}{1.5}
  \end{lscinst}
\end{Lsc}
```

The order of the finalising commands of composite commands and cuts *is* significant. They are drawn immediately, the one occurring first is drawn first.

14 Technicalities

The `lsc.sty` package is in particular used to describe the semantics of LSCs which depends on the notion of atoms and cuts, i.e. sets of atoms. There commands introduced in this section are mainly intended to be able to illustrate such descriptions. For example, to show the position of a cut.

14.1 Atoms

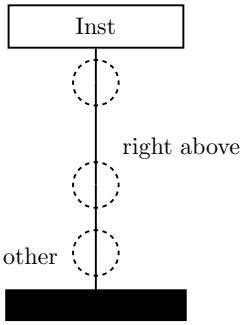
```
\lscAtom{id}
\lscLabelAtomAt{angle}{id}{label}
\lscLabelAtom[ra|rb|la|lb]{id}{label}
```

The `\lscAtom` command marks the (atom(s) at the) current location with a dashed circle and equips it with an identity *id*. This identity may then be used to label the atom mark.

The command `\lscLabelAtomAt` puts the *label* at position *angle* relative to the atom mark with identity *id*.

The command `\lscLabelAtom` is a shortcut for the most common angles. The default is -45 degrees ('`ra`' (right above)), the other options are -135

degrees (‘rb’ (right below)), 45 degrees (‘la’ (left above)), and 135 degrees (‘lb’ (left below)).



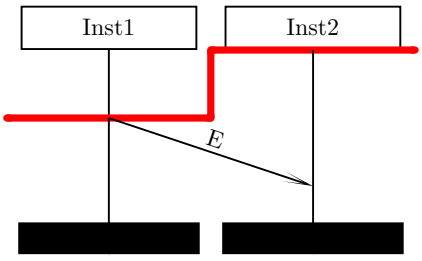

```
\begin{Lsc}{none}{2}{4}
  \begin{lscinst}{Inst}
    \lscLine{hot}{0.5}
    \lscAtom{aa}
    \lscLine{hot}{1.5}
    \lscAtom{ab}
    \lscLine{hot}{1}
    \lscAtom{ac}
    \lscLine{hot}{1}
  \end{lscinst}
  \lscLabelAtom{ab}{right above}
  \lscLabelAtomAt{93}{ac}{other}
\end{Lsc}
```

14.2 Cuts

```
\lscCut
\lscSetCutColour{colour}
\lscSetCutLinestyle{linestyle}
\lscSetCutWidth{length}
```

The `\lscCut` atomic command draws a vertical line at the current position indicating the location of the cut on the current instance line. The line is connected to the last existing cut on the left. There shall not be more than one `\lscCut` command per instance line.

The command `\lscSetCutColour` sets the colour to the colour name *colour* (default: red), `\lscSetCutLinestyle` sets the line style to *linestyle* (default: solid), and `\lscSetCutWidth` sets the width to *length* (default: 3pt).

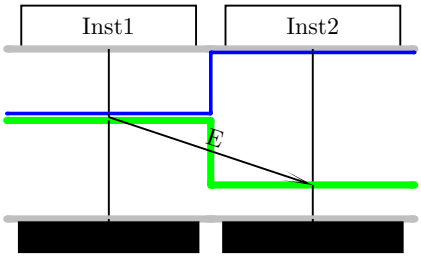



```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}%
    \lscCut%
    \lscAsynSnd{e}%
    \lscLine{hot}{2}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscCut%
    \lscLine{hot}{1}%
    \lscAsynRcv{e}%
      \psset{linecolor=red}
    \end{coregion}
    \lscLine{hot}{1.5}%
  \end{lscinst}
  \lscAsyn{e}{hot}{E}%
\end{Lsc}}
```

```
\lscNamedCut{id}
\lscSetNamedCutColour{id}{colour}
\lscSetNamedCutLinestyle{id}{linestyle}
\lscSetNamedCutWidth{id}{length}
```

To allow multiple cuts in a single LSC, there is a corresponding set of commands for named cuts. They take an *id* as first parameter.

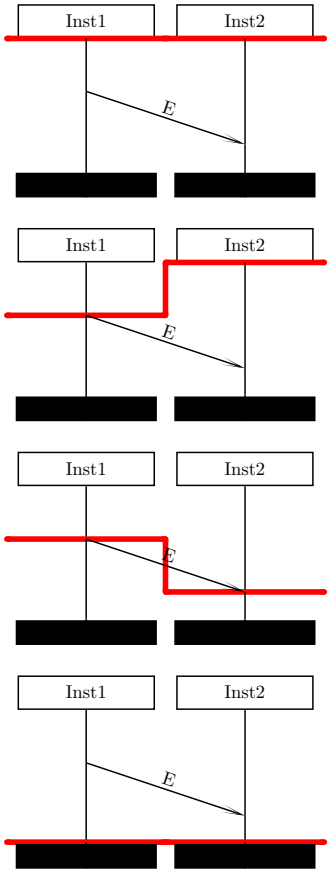
If not set explicitly, the colour, line style, and width of named cuts is inherited from the general cut colour, line style, and width as set by `\lscSetCutColour`, `\lscSetCutLinestyle`, and `\lscSetCutWidth`.




```
\lscSetCutColour{lightgray}
\lscSetNamedCutWidth{Xb}{1.5pt}
\lscSetNamedCutColour{Xc}{green}
\lscSetNamedCutColour{Xb}{blue}
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscNamedCut{Xa}%
    \lscLine{hot}{0.95}%
    \lscNamedCut{Xb}%
    \lscLine{hot}{0.05}
    \lscAsynSnd{e}%
    \lscLine{hot}{0.05}
    \lscNamedCut{Xc}%
    \lscLine{hot}{1.45}%
    \lscNamedCut{Xd}%
    \lscLine{hot}{0.5}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscNamedCut{Xa}%
    \lscLine{hot}{0.05}%
    \lscNamedCut{Xb}%
    \lscLine{hot}{1.95}%
    \lscNamedCut{Xc}%
    \lscAsynRcv{e}%
    \lscLine{hot}{0.5}%
    \lscNamedCut{Xd}%
    \lscLine{hot}{0.5}%
  \end{lscinst}
  \lscAsyn{e}{hot}{E}%
\end{Lsc}
```

To avoid cuts to hide each other, one will typically add very short instance line segments in between as shown in the example above.

Using line style ‘none’ one can easily obtain an animation effect as shown in the following example.



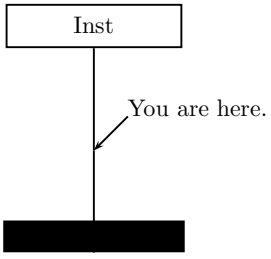

```
\newcommand{\lscanim}[1]{%
  \lscSetCutLinestyle{none}%
  \lscSetNamedCutLinestyle{#1}{solid}%
  \begin{Lsc}{none}{2}{3}
    \begin{lscinst}{Inst1}
      \lscNamedCut{Xa}%
      \lscLine{hot}{1}%
      \lscNamedCut{Xb}%
      \lscNamedCut{Xc}%
      \lscAsynSnd{e}%
      \lscLine{hot}{1.5}%
      \lscNamedCut{Xd}%
      \lscLine{hot}{0.5}%
    \end{lscinst}
    \begin{lscinst}{Inst2}
      \lscNamedCut{Xa}%
      \lscNamedCut{Xb}%
      \lscLine{hot}{2}%
      \lscNamedCut{Xc}%
      \lscAsynRcv{e}%
      \lscLine{hot}{0.5}%
      \lscNamedCut{Xd}%
      \lscLine{hot}{0.5}%
    \end{lscinst}
    %
    \lscAsyn{e}{hot}{E}%
  \end{Lsc}}
\lscanim{Xa}\lscanim{Xb}
\lscanim{Xc}\lscanim{Xd}
```

14.3 Putting Stuff at the Current Location

`\lscPut{`*stuff*`}`

The `\lscPut` command typesets *stuff* at the current location. This is useful to put additional labels or elements for which there is no explicit support.

We use `\lscPut`, for example, to draw the dashed frames indicating the location dimensions in Figure 2.




```
\begin{Lsc}{none}{1}{3}
  \begin{lscinst}{Inst}
    \lscLine{hot}{1.5}
    \lscPut{%
      \psline{<-}(0.5,0.5)%
      \rput[lb](0.5,0.5){You are here.}}
    \lscLine{hot}{1.5}
  \end{lscinst}
\end{Lsc}
```

15 Typesetting Play-Engine LSCs

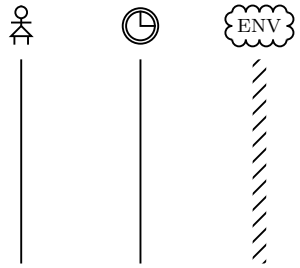
As mentioned in the introduction, `lsc.sty` in general renders LSC in the style of [Klo03]. But it provides a couple of commands which support the typesetting of LSCs in the play-engine style.

In cases where the play-engine style is not supported, it may be sufficient to modify the existing elements, for example, by manually adding an hourglass to an action box to represent the saving of a timer.

15.1 Actors, Clocks, and Environments

```
\lscActor
\lscClock
\lscEnv
```

The `\lscActor`, `\lscClock`, and `\lscEnv` draw the symbols which are used to mark the actor, clock, and environment instance line. They are best used as the label of a '`noboxSD`' instance.



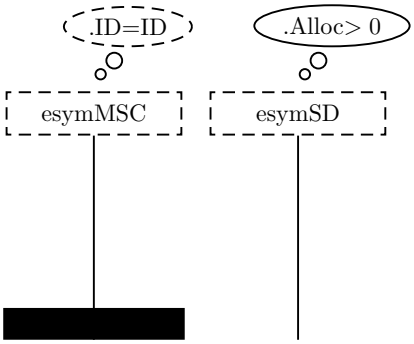

```
\begin{Lsc}{none}{3}{3}
  \begin{lscinst}[noboxSD]{\lscActor}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}[noboxSD]{\lscClock}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}[noboxSD]{\lscEnv}
    \lscEnvLine{cold}{3}
  \end{lscinst}
\end{Lsc}
```

15.2 Existential Symbolic Instances

`\lscBalloon{existential|universal}{`*stuff*`}`

The `\lscBalloon` command draws think balloons that are used to provide binding expressions of symbolic instances.

They have to be explicitly positioned using, e.g., `\lscPut` and one has to care for vertical space before the LSC as they are not considered in the computation of the boundary box. That is, symbolic instances can at best be called semi-supported.

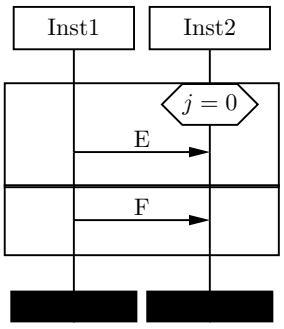



```
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}[esymMSC]{esymMSC}
    \lscPut{\rput(0,0.9){%
      \lscBalloon%
        {existential}{.ID$=$ID}}}
    \lscLine{hot}{3}
  \end{lscinst}
  \begin{lscinst}[esymSD]{esymSD}
    \lscPut{\rput(0,0.9){%
      \lscBalloon%
        {universal}{.Alloc$> 0$}}}
    \lscLine{hot}{3}
  \end{lscinst}
\end{Lsc}
```

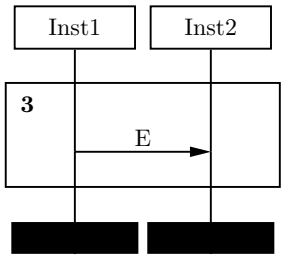
The combination on the right above is not a well-formed LSC instance. It is only used to demonstrate both kinds of loops in the same figure.

15.3 Branching, Subcharts, and Loops

Both branching and loops are based on subcharts. Subcharts are not directly supported but can be simulated by putting frames using `\lscPut`.




```
\begin{Lsc}{none}{3}{4}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1.5}\lscInstSnd{e}
    \lscLine{hot}{1}\lscInstSnd{f}
    \lscLine{hot}{1.5}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{0.5}
    \lscCond{hot}{$j = 0$}
    \lscLine{hot}{1}\lscInstRcv{e}
    \lscLine{hot}{0.5}
    \lscPut{\psframe[dimen=inner]%
     (-1.5\lscLocationWidth,0)%
     (0.5\lscLocationWidth,1.5\lscLocationHeight)}
    \lscLine{hot}{0.5}\lscInstRcv{f}
    \lscLine{hot}{0.5}
    \lscPut{\psframe[dimen=inner]%
     (-1.5\lscLocationWidth,0)%
     (0.5\lscLocationWidth,1\lscLocationHeight)}
    \lscLine{hot}{1}
  \end{lscinst}
  \lscInst{e}{hot}{E}\lscInst{f}{hot}{F}
\end{Lsc}
```

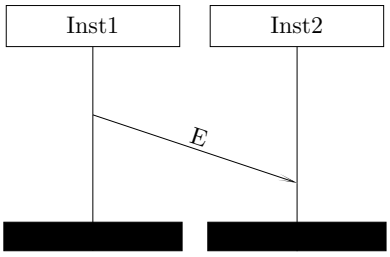



```
\begin{Lsc}{none}{3}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1.5}\lscInstSnd{e}
    \lscLine{hot}{1.5}
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{1.5}\lscInstRcv{e}
    \lscLine{hot}{0.5}
    \lscPut{\psframe[dimen=inner]%
       (-1.5\lscLocationWidth,0)%
       (.5\lscLocationWidth,1.5\lscLocationHeight)
      \rput[lt](-1.5\lscLocationWidth,%
       1.5\lscLocationHeight){%
         \rput(0.3,-0.3){\textbf{3}}}}
    \lscLine{hot}{1}
  \end{lscinst}
  \lscInst{e}{hot}{E}
\end{Lsc}
```

16 Tricks and Tweaks

16.1 Linewidth

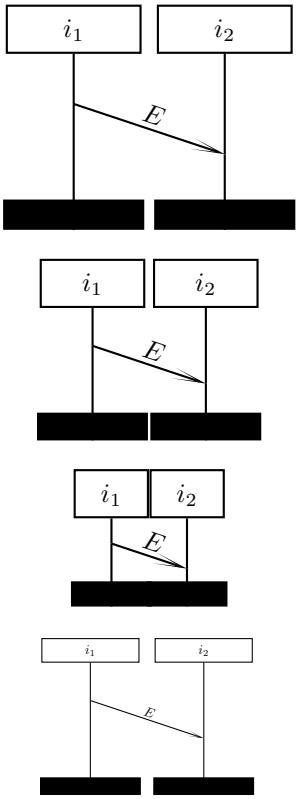
All parts of the LSC (except for cuts) are rendered with the current `pstricks` linewidth. That it, if the lines appear too fat they can be globally be changed using `\psset`.




```
\psset{linewidth=0.25pt}
\begin{Lsc}{none}{2}{3}
  \begin{lscinst}{Inst1}
    \lscLine{hot}{1}%
    \lscAsynSnd{e}%
    \lscLine{hot}{2}%
  \end{lscinst}
  \begin{lscinst}{Inst2}
    \lscLine{hot}{2}%
    \lscAsynRcv{e}%
    \lscLine{hot}{1}%
  \end{lscinst}
  \lscAsyn{e}{hot}{E}%
\end{Lsc}
```

16.2 Scaling

LSCs can be scaled by changing the width and height of locations (cf. Section 4.3) keeping the size of text. To scale the LSC including all annotations, the `FullLsc` and `Lsc` environments can be passed to a `\scalebox` command.




```
\newcommand{\scalex}{%
  \begin{Lsc}{none}{2}{3.5}
    \begin{lscinst}{$i_1$}
      \lscLine{hot}{1}%
      \lscAsynSnd{e}%
      \lscLine{hot}{2.5}%
    \end{lscinst}
    \begin{lscinst}{$i_2$}
      \lscLine{hot}{2}%
      \lscAsynRcv{e}%
      \lscLine{hot}{1.5}%
    \end{lscinst}
    \lscAsyn{e}{hot}{$E$}%
  \end{Lsc}

{\pssetlength{\lscLocationWidth}{2}
\pssetlength{\lscLocationHeight}{0.666}
\scalex}%

{\pssetlength{\lscLocationWidth}{1.5}
\pssetlength{\lscLocationHeight}{0.5}
\scalex}%

{\pssetlength{\lscLocationWidth}{1}
\pssetlength{\lscLocationHeight}{0.333}
\scalex}%

\scalebox{0.5}{\scalex}
```

Index

References

[DH01] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, July 2001.

[DW05] Werner Damm and Bernd Westphal. Live and let die: LSC-based verification of UML-models. *Science of of Computer Programming*, 55(1–3):117–159, March 2005.

[HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[IT99] ITU-T. *ITU-T Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-T, 1999.

[Klo03] Jochen Klose. *Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 2003.

[KW02] Jochen Klose and Bernd Westphal. Relating LSC specifications to UML models. In Hartmut Ehrig and Martin Grosse-Rhode, editors, *Proceedings of the Workshop Integration of Software Specification Techniques, INT'02*, pages 130–137, April 2002.

[MB01] Sjouke Mauw and Victor Bos. Drawing Message Sequence Charts with LaTeX. *TUGBoat*, 22(1-2):87–92, March/June 2001.

License