Introduction to XClips

Introduction to XClips

☺ *Agent Zhang* ☺

(章亦春)

2006.11

XClips

- *Expert system* programming language

XClips是一种专家系统编程语言

XClips

➥ **Targeting NASA's *CLIPS***

XClips 以美国航空航天局的 CLIPS 作为目标平台

XClips

➡ *Born* in my VRG project but is also a *general-purpose* language

XClips 诞生于我的 VRG 项目但同时也是通用目的的语言。

Features of XClips

Features of XClips

- *Prolog*-style *syntax* for rules and facts

Features of XClips

- *Prolog* -style *syntax* for rules and facts
- *Perl 6* -style *user-defined* operators

Features of XClips

- *Prolog*-style *syntax* for rules and facts
- *Perl 6*-style *user-defined* operators
- *C*-style *file inclusion* and automatic reinclusion elimination

Features of XClips

- *Prolog*-style *syntax* for rules and facts
- *Perl 6*-style *user-defined* operators
- *C*-style *file inclusion* and automatic reinclusion elimination
- *Good* access to the *CLIPS* *semantics*

Features of XClips

♡ *Prolog* -style *syntax* for rules and facts

♡ *Perl 6* -style *user-defined* operators

♡ *C* -style *file inclusion* and automatic reinclusion elimination

♡ *Good* access to the *CLIPS* *semantics*

♡ Rule *coverage* *testing* support

Features of XClips

- *Prolog* -style *syntax* for rules and facts
- *Perl 6* -style *user-defined* operators
- *C* -style *file inclusion* and automatic reinclusion elimination
- *Good* access to the *CLIPS* *semantics*
- Rule *coverage* *testing* support
- *Inferencing* process *visualization* support

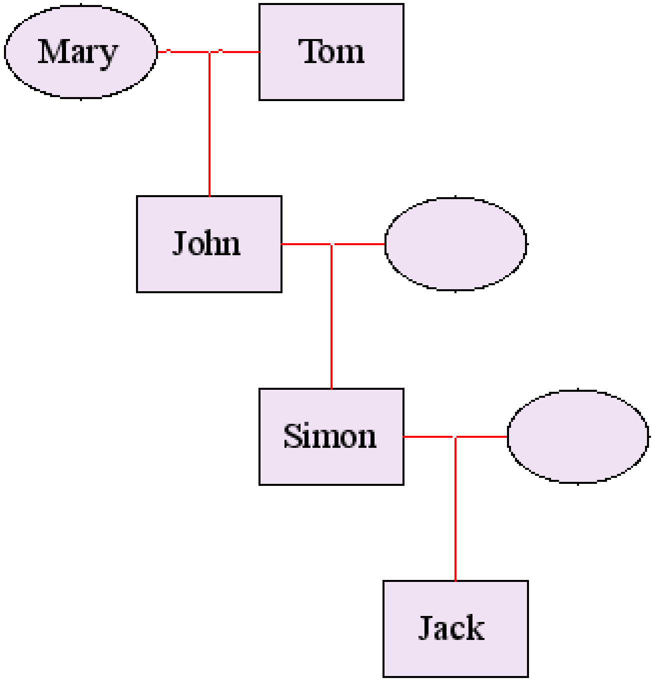
Features of XClips

- *Prolog* -style *syntax* for rules and facts
- *Perl 6* -style *user-defined* operators
- *C* -style *file inclusion* and automatic reinclusion elimination
- *Good* access to the *CLIPS* *semantics*
- Rule *coverage* *testing* support
- *Inferencing* process *visualization* support
- *Modular* programming

Case #1 : Reasoning on *family* trees

♨

案例 #1: 家族树上的推理


Mary
Tom
John
Simon
Jack

☺ *Defining* XClips facts

定义 XClips 事实

```
mother(Mary, John).
father(John, Simon).
father(Simon, Jack).
father(Tom, John).
```

Unlike Prolog, symbols with leading a capital letter or underscore are *not* variables.

与 Prolog 不同的是，以大写字母或下划线起始的符号并不是变量。

☺ *Defining* XClips rules

定义 XClips 规则

```
mother( ?A ,  ?B ); father( ?A ,  ?B ) =>
          ancestor( ?A ,  ?B ).

ancestor( ?A ,  ?B ), ancestor( ?B ,  ?C ) =>
          ancestor( ?A ,  ?C ).

ancestor(Tom,  ?X ) =>  printout ( t ,  ?X ,  crlf ).
```

Unlike Prolog, the *rules* are expressed in the *forward-chaining* style.

与 Prolog 不同的是，规则被表达为正向链风格

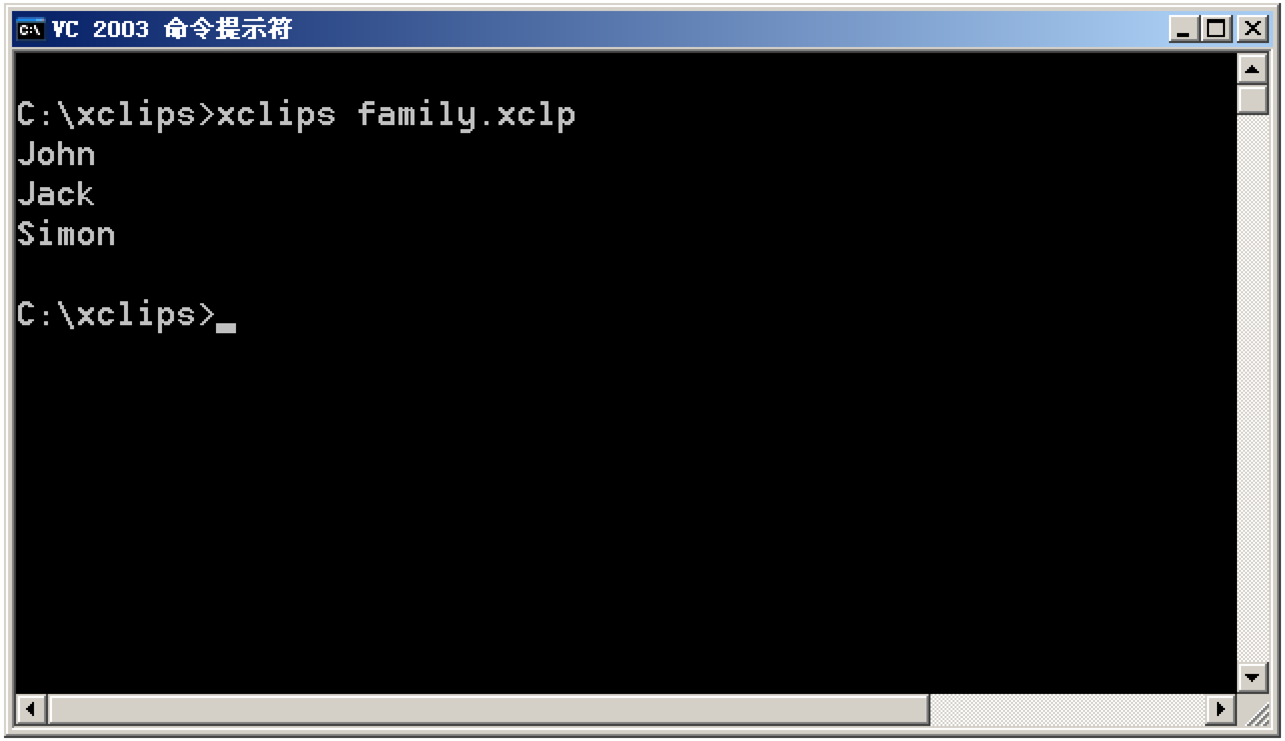
Unlike Prolog, the *variables* are specified by the ? sigil.

与 Prolog 不同的是，变量是通过问号这个魔法标记来指示的。

Let's put *rules* and *facts* together.

让我们把规则和事实放到一起。

```
 /* family.xclp */
mother( Mary ,  John ).
father( John ,  Simon ).
father( Simon ,  Jack ).
father( Tom ,  John ).
mother( ?A ,  ?B ); father( ?A ,  ?B ) =>
          ancestor( ?A ,  ?B ).
ancestor( ?A ,  ?B ), ancestor( ?B ,  ?C ) =>
          ancestor( ?A ,  ?C ).
ancestor( Tom ,  ?X ) =>  printout ( t ,  ?X ,  crlf ).
```



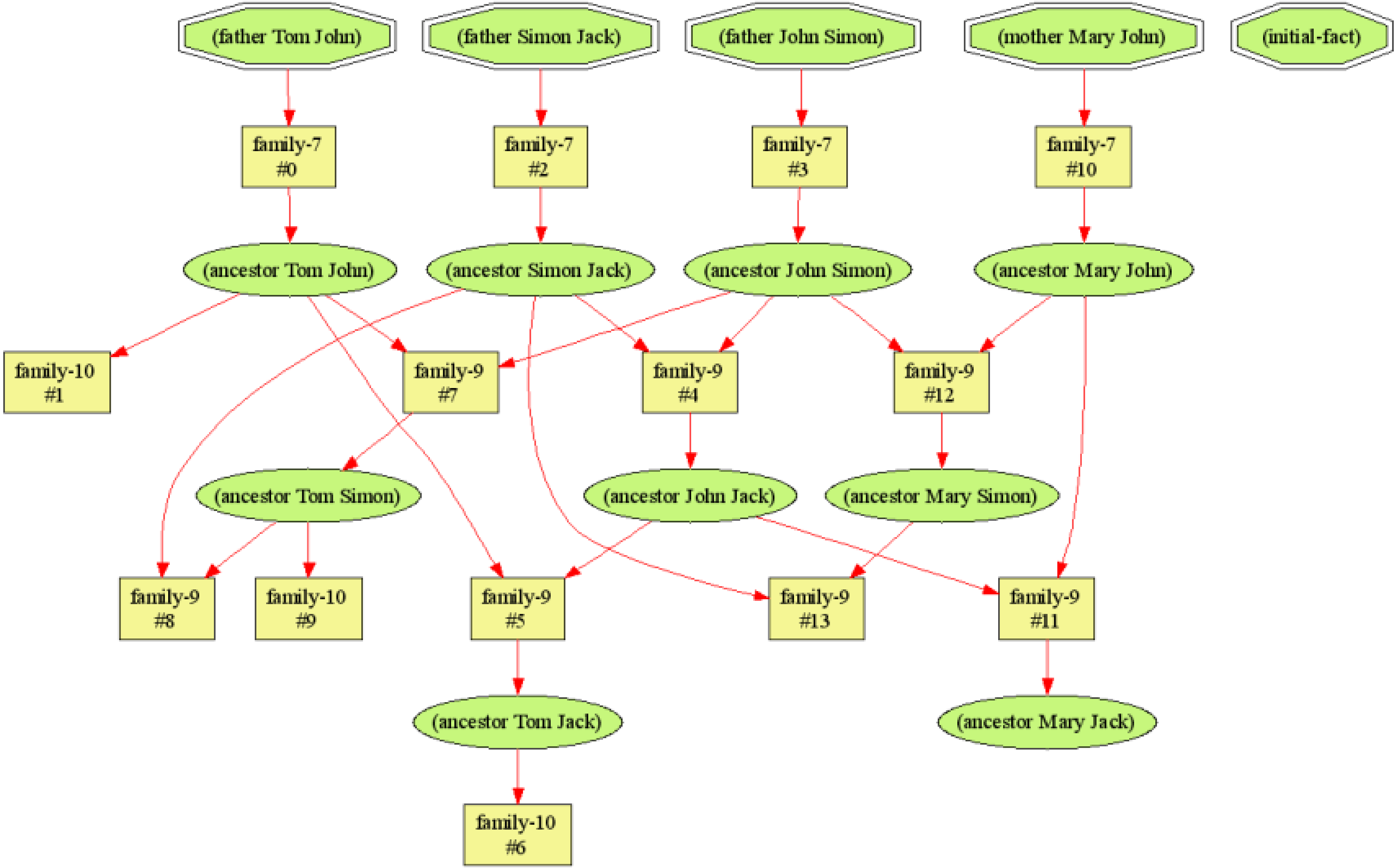
```
C:\xclips>xclips family.xclp
John
Jack
Simon

C:\xclips>_
```

```
 C:\xclips>xclips family.xclp
John
Jack
Simon

 C:\xclips>
```

☺ Generating *inferencing flowcharts*

生成推理流程图

```
C:\xclips>xclips -d family.xclp
 generating a.png...

C:\xclips>
```


(father Tom John)
(father Simon Jack)
(father John Simon)
(mother Mary John)
(initial-fact)
family-7 #0
family-7 #2
family-7 #3
family-7 #10
(ancestor Tom John)
(ancestor Simon Jack)
(ancestor John Simon)
(ancestor Mary John)
family-10 #1
family-9 #7
family-9 #4
family-9 #12
(ancestor Tom Simon)
(ancestor John Jack)
(ancestor Mary Simon)
family-9 #8
family-10 #9
family-9 #5
family-9 #13
family-9 #11
(ancestor Tom Jack)
(ancestor Mary Jack)
family-10 #6

- ☺ Generating *rule coverage* reports

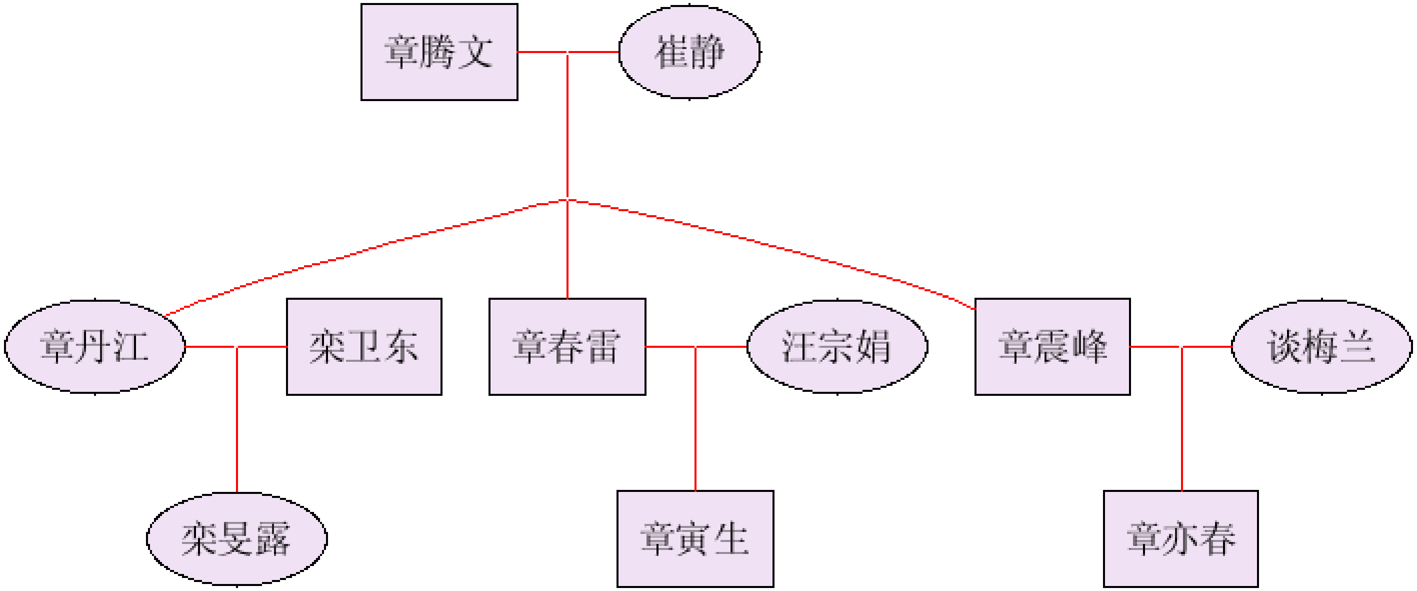
生成规则覆盖报告

```
C:\xclips>xclips -d family.xclp
 generating a.png...
C:\xclips>clips-cover
 ---------------
 Rule       Count
 ---------------
 family-10  3
 family-8   4
 family-9   7

 For total 100.00% of the rules have been fired.
```

Let's study a *real-world* pedigree.

♨

让我们研究一下真实世界的家谱。


章腾文
崔静
章丹江
栾卫东
章春雷
汪宗娟
章震峰
谈梅兰
栾旻露
章寅生
章亦春

☺ Defining the facts and goals

定义事实和目标

```
/* myfamily.xclp */
father("章腾文", "章丹江").
father("章腾文", "章春雷").
father("章腾文", "章震峰").
couple("章腾文", "崔静").
mother("章丹江", "栾昱露").
father("栾卫东", "栾昱露").
mother("汪宗娟", "章寅生").
father("章春雷", "章寅生").
mother("谈梅兰", "章亦春").
father("章震峰", "章亦春").
/* goal */
ancestor( ?elder , "章亦春") =>
    printout ( t , "Found ", ?elder , crlf ).
```

☆ Double-quotes are *required* here because Chinese glyphs are *not* valid XClips symbols

这儿的双引号是不能省略的，
因为汉字不是合法的XClips符号。

- ☺ Defining rules for *ancestor* and *couple*

为 ancestor 和 couple 定义规则

```
/* pedigree-rules.xclp */

mother( ?a ,  ?b ); father( ?a ,  ?b ) => ancestor( ?a ,  ?b ).
ancestor( ?a ,  ?b ), ancestor( ?b ,  ?c ) => ancestor( ?a ,  ?c ).
couple( ?a ,  ?b ), father( ?a ,  ?c ) => mother( ?b ,  ?c ).
couple( ?a ,  ?b ), mother( ?a ,  ?c ) => father( ?b ,  ?c ).
```

```
C:\xclips> xclips myfamily.xclp pedigree-rules.xclp
Found 章震峰
Found 谈梅兰
Found 崔静
Found 章腾文
```

☆ This time, we put reusable rules into a *separate* file.

这一回，我们将可复用的规则
放进了单独的一个文件。

☺ We can precompile pedigree-rules.xclp to CLIPS source in order to *save* compilation time.

我们可以将pedigree-rules.xclp预编译成CLIPS源代码，以便节约编译时间。

```
C:\xclips>xclips -c pedigree-rules.xclp

C:\xclips>xclips myfamily.xclp pedigree-rules.clp
 Found 章震峰
 Found 谈梅兰
 Found 崔静
 Found 章腾文
```

☆ For this specific example, *precompilation* can save 18% of the total time.

对于这个例子而言，预编译可以节约 18% 的总时间。

While debugging, the -v option may come to handy.
It instructs *xclips* to print the underlying CLIPS sessions

当调试的时候，-v 选项可能是很方便的。

```
C:xclips> xclips  -v  myfamily.xclp pedigree-rules.clp
         CLIPS (V6.24 06/15/06)
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (father "章腾文"  "章丹江")
==> f-2     (father "章腾文"  "章春雷")
==> f-3     (father "章腾文"  "章震峰")
==> f-4     (couple "章腾文"  "崔静")
==> f-5     (mother "章丹江"  "栾昱露")
==> f-6     (father "栾卫东"  "栾昱露")
...
```

```
...
CLIPS> (rules *)
MAIN:
   myfamily-17
   pedigree-rules-1
   pedigree-rules-2
   pedigree-rules-3
   pedigree-rules-4
For a total of 5 defrules.
CLIPS> (run)
FIRE    1 pedigree-rules-1: f-10
==> f-11    (ancestor "章震峰" "章亦春")
FIRE    2 myfamily-17: f-11
Found 章震峰
...
```

☺ Generating *rule coverage* reports (again)

生成规则覆盖报告
（再一次地）

```
C:\xclips>clips-cover -d
C:\xclips> xclips -d myfamily.xclp pedigree-rules.clp
generating a.png...
C:\xclips>clips-cover
----------------------
Rule             Count
----------------------
pedigree-rules-6  0
pedigree-rules-5  3
myfamily-17       4
pedigree-rules-4  6
pedigree-rules-3  12

For total 80.00% of the rules have been fired.
```

The rule that has *never* been fired:

➥ pedigree-rules-6

从未触发过的规则：predigree-rules-6

The rule that has *never* been fired:

➡ pedigree-rules-6

从未触发过的规则：predigree-rules-6

(i.e. the rule at line 6 of the file pedigree-rules.xclp)

即 pedigree-rules.xclp 文件的第 6 行上的规则

```
/* pedigree-rules.xclp */

mother(?a, ?b); father(?a, ?b) => ancestor(?a, ?b).
ancestor(?a, ?b), ancestor(?b, ?c) => ancestor(?a, ?c).
couple(?a, ?b), father(?a, ?c) => mother(?b, ?c).
couple(?a, ?b), mother(?a, ?c) => father(?b, ?c).
```

Case #2 : Reasoning in Geometry

♨

案例 #2: 几何学中的推理

If l, m, and n are all lines in 3-space, and $l \parallel m$, $m \parallel n$, then $l \parallel n$.

如果 l, m, 和 n 都是三维空间中的直线，且 l ∥ m, m ∥ n，则有 l ∥ n.

☆ Let's translate this theorem to an XClips rule.

让我们将这个定理翻译成 XClips 规则。

```
 /* geometry.xclp */

line( ?l ), line( ?m ), line( ?n ),
parallel( ?l , ?m ), parallel( ?m , ?n ), ?l \= ?n ,
    => parallel( ?l , ?n ).
```

☹ Oh, it looks *far* too verbose…

哦，它看上去太罗嗦了……

☆ Let's add some *syntax sugar* by defining some operators of our own!

让我们通过定义一些我们自己的运算符来添加一些“记法糖”！

```
/* geometry.xclp */

prefix :<|>   line
infix  :<//>  parallel

|?l , |?m , |?n ,
?l // ?m , ?m // ?n , ?l \= ?n
    => ?l // ?n .
```

Perl 6 style operator definition syntax!

Perl 6 风格的运算符定义记法！

XClips currently supports the following operator categories :

XClips currently supports the following operator categories :

```
prefix :<X>
```

XClips currently supports the following operator categories :

`prefix :<X>`

`infix :<X>`

XClips currently supports the following operator categories :

```
prefix :<X>
infix :<X>
infix_prefix :<X>
```

XClips currently supports the following operator categories :

```
prefix :<X>
infix :<X>
infix_prefix :<X>
infix_circumfix :<X Y>
```

- ☺ Let's define some facts so as to *test* it.

让我们来定义一些事实，以便测试它一下。

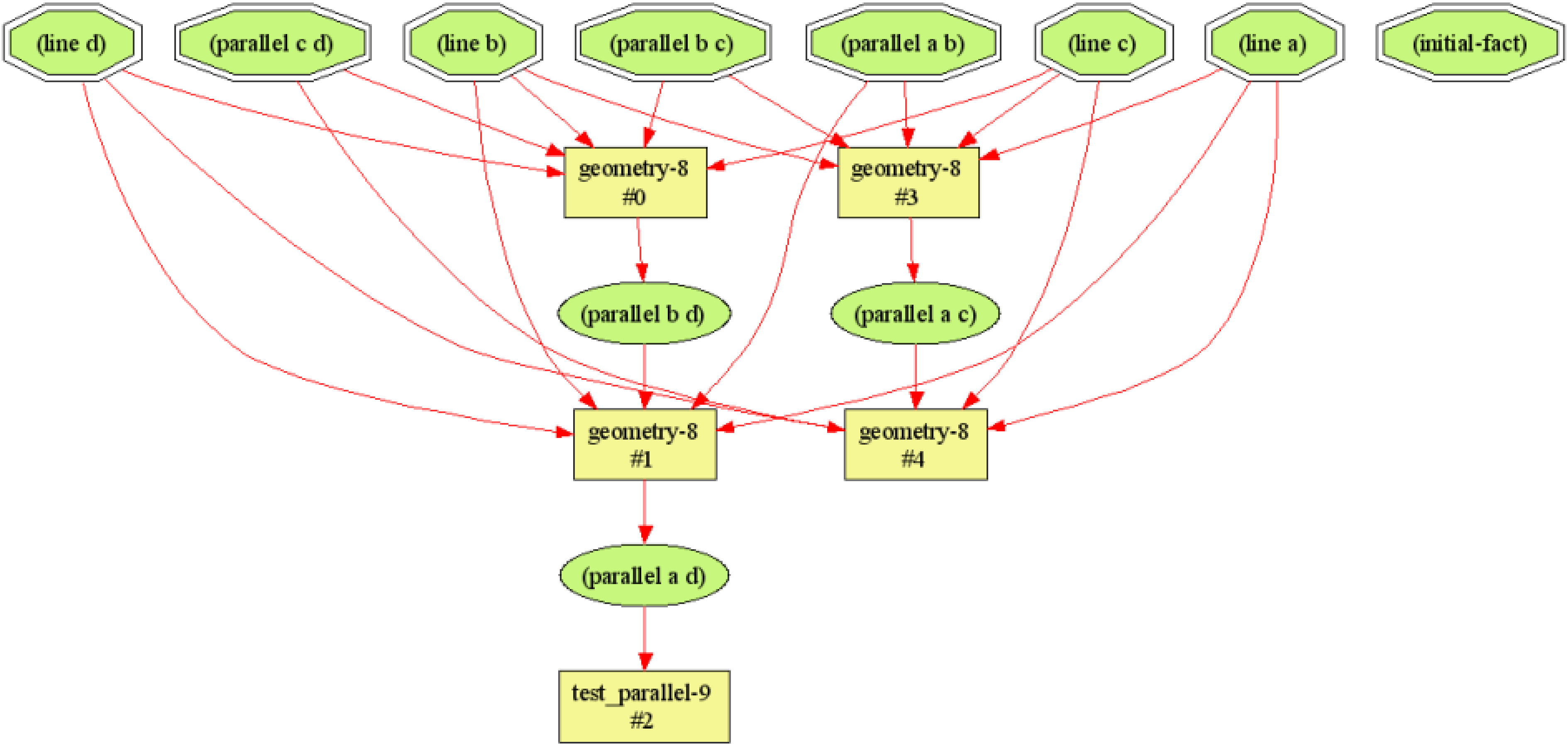
```
/* test_parallel.xclp */
prefix :<|>  line
infix :<//>  parallel

|a,  |b,  |c,  |d.
 a // b, b // c, c // d.

/* the goal */
 a // d => printout ( t , "Yes.",  crlf ).
```

```
C:\xclips> xclips test_parallel.xclp geometry.xclp
Yes.
```

☺ Generate the *inferencing flowchart*:

生成推理流程图

```
C:\xclips> xclips  -d  test_parallel.xclp geometry.xclp
generating  a.png ...
```


(line d)
(parallel c d)
(line b)
(parallel b c)
(parallel a b)
(line c)
(line a)
(initial-fact)
geometry-8 #0
geometry-8 #3
(parallel b d)
(parallel a c)
geometry-8 #1
geometry-8 #4
(parallel a d)
test_parallel-9 #2

It's meaningful to *share* the *operator definitions* among facts and rules.

在事实与规则之间共享运算符定义是有意义的。

```
/* sugar.xclp */

prefix:<|>  line
infix:<//>  parallel
```

```
/* geometry.xclp */

include  "sugar.xclp"

|?l , |?m , |?n ,
?l  //  ?m , ?m  //  ?n , ?l  \=  ?n
    =>  ?l  //  ?n .
```

```
/* test_parallel.xclp */

include  "sugar.xclp"

|a, |b, |c, |d.
 a // b, b // c, c // d.

/* the goal */
 a // d => printout(t, "Yes.", crlf).
```

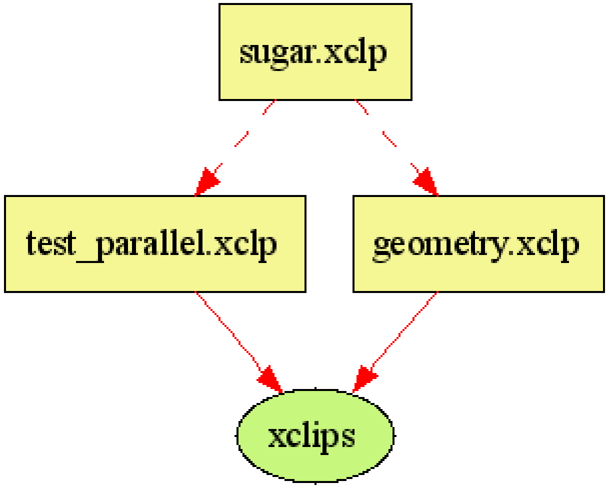
☆ *C*-style file inclusion !

C 风格的文件包含！

```
C:\xclips> xclips test_parallel.xclp geometry.xclp
Yes.
```

Never feed *included* files to xclips.

永远不要将被包含的文件喂给 xclips.


sugar.xclp
test_parallel.xclp
geometry.xclp
xclips

☆ *Unlike* C, XClips will *automatically* prevent a file to be included *more* than once. So you don't need to worry about that.

与 C 不同的是，XClips 会自动避免一个文件被重复包含多次，因此你不必担心这个问题。

☺ Some more examples for *user-defined operators*

更多的用户自定义运算符的示例……

```
infix :<T>   perpendicular

 a T b, b T c.

/* equivalent to:
   perpendicular(a, b), perpendicular(b, c). */
```

```
infix_prefix :<~>  not_ &

a ~// b, a ~T b.

/* equivalent to:
   not_parallel(a, b), not_perpendicular(b, c). */
```

```
infix_circumfix :<[ ]>  space-relation

 a [//] b, a [T] b.

/* equivalent to:
   space-relation(parallel, a, b),
   space-relation(perpendicular, b, c). */
```

```
infix_circumfix :<< >>  vector-relation

?a <?R> ?b => ?a <?R> ?b .

/* equivalent to:
   vector-relation(?R, ?a, ?b) =>
   vector-relation(?R, ?a, ?b). */
```

☺ I'm adding new operator categories like

☺ I'm adding new operator categories like

```
postfix:<X>,
```

☺ I'm adding new operator categories like

```
postfix:<X>,
circumfix:<X Y>,
```

- I'm adding new operator categories like

```
postfix :<X>,
circumfix :<X Y>,
infix_postfix :<X>,
```

☺ I'm adding new operator categories like

```
postfix:<X>,
circumfix:<X Y>,
infix_postfix:<X>,
```

and even *more*...

☆ *XClips* tries to give you the *power* of notation .

XClips 努力带给你记法的威力。

☆ *Modular* programming support in XClips

XClips 的模块化编程支持

```
define defmodule(FOO,
    import(BAR, deftemplate, ?ALL),
    import(MAIN, deftemplate, initial-fact)
)

module FOO.

/* rules and facts go here */
```

☺ Yes, it's just a *thin* wrapper around the CLIPS syntx. I'll *abstract* this to some more elegant form in the future.

是的，它仅仅是对 CLIPS 记法的一层薄薄的包裹。
未来我将把它抽象为更为优雅的形式。

Thank you!

☺